AD-A105 381 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH F/G 9/2
TIME-EXTENDED PETRI NETS.(U)
AUG 79 F B BERLIN
UNCLASSIFIED AFIT-CI-79-205T NL

1 OF 2
AD
A105381

AD A105381

14

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AFIT-CI-

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 79-205T	2. GOVT ACCESSION NO. AD A105381	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Time-Extended Petri Nets.		5. TYPE OF REPORT & PERIOD COVERED THESIS/~~DISSERTATION~~
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Frank Brett/Berlin		8. CONTRACT OR GRANT NUMBER(s) (9) Master's Thesis
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: The University of Texas at Austin		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS AFIT/NR WPAFB OH 45433		12. REPORT DATE (11) August 1979
		13. NUMBER OF PAGES 143
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (12) 164		15. SECURITY CLASS. (of this report) UNCLASS
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

(1) LEVEL II

DTIC SELECTE OCT 9 1981 D

S H

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

23 JUN 1981 [illegible]

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-17

FREDRIC C. LYNCH, Major, USAF
Director of Public Affairs
Air Force Institute of Technology (ATC)
Wright-Patterson AFB, OH 45433

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ATTACHED

DTIC FILE COPY

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

012200

ABSTRACT

This thesis develops a new computer performance evaluation structure called the time-extended Petri net which retains logical synchronization and concurrency characteristics of systems. Cost effectiveness is one of the important considerations together with an evaluation of how it works. The overall objective is to obtain a model to determine the automatic data processing dollar's efficiency.

~~14~~ 1057

TIME-EXTENDED PETRI NETS

To my parents,

William A. and Loraine S. Berlin

and to my Lord,

Jesus, the Christ

"I am the vine, you are the branches;
he who abides in Me, and I in him, he bears much fruit;
for apart from Me you can do nothing."

— Jesus Christ (John 15:5)

TIME-EXTENDED PETRI NETS

by

FRANK BRETT BERLIN, B.S.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF ARTS

THE UNIVERSITY OF TEXAS AT AUSTIN

August 1979

PREFACE

Computer performance evaluation (CPE) is primarily a matter of economics. Whereas some evaluate a modeling methodology in theoretical terms, the CPE analyst must ask questions of more direct application: "Does it work?"; "How much will it cost?" CPE tends to be a pragmatic discipline aimed at having a direct impact upon the Automatic Data Processing (ADP) dollar's efficiency.

The relationship between ADP and CPE costs has fostered growing commitment in government and industry to the development of effective CPE tools. This commitment has resulted in CPE's transformation into a multimillion dollar industry and an important computer science research discipline. Not many years ago, performance measurement and analysis was a "seat of the pants" endeavor to all but a small coterie of experts. Today, sophisticated packaged tools exist which allow the trained technician to easily and accurately model and evaluate many aspects of a computer system's performance.

Despite the economic significance and considerable progress which the performance evaluation community has enjoyed during recent years, CPE researchers have still had difficulty responding to the challenges posed by new system architectures and operating philosophies: CPE tools are needed which can be used to develop accurate deterministic models of parallel systems at a small cost relative to the system cost. Current CPE tools do not adequately meet these challenges. For example, as this Thesis points out, the queueing network model is a user-oriented, inexpensive modeling tool. Yet, the basic assumptions of the model make it an inadequate tool for studying many deterministic parallel systems. When these inadequacies become important factors in a study, the CPE analyst must either accept an approximate solution or use other more expensive and more cumbersome modeling methods, such as discrete simulation.

Because of the importance of parallel systems within the ADP user community, there is a strong need for a cost-effective, data-driven performance modeling methodology which can faithfully represent deterministic behavior, process blocking, and the holding of multiple resources by a single process. At least one study [Browne, et. al., 1973] has shown that currently available tools are inadequate due to their expense, difficulty of use, or inherent modeling limitations; in some cases these limitations are significant constraints.

The objective of this Thesis is to introduce a modeling methodology which meets the needs described above. While it would be naive, if not presumptuous, to imply an ultimate solution, the methodology is demonstrated to be useful in analyzing the performance of a sophisticated disk subsystem that can only be approximated by queueing networks. The capabilities of the model demonstrated by this example problem seem to indicate that the Time-Extended Petri Net (TEPN) offers a fresh and useful tool for cost-effective CPE.

Acknowledgements

As is usually the case with a project of the magnitude of this Thesis, it is very difficult to adequately thank those who have provided ideas, friendship and encouragement, and who have in many ways made the Thesis possible. If I were to mention all those who deserve some credit for this Thesis, it would take several pages. However, there are a few individuals who, because of their special investment in this project, warrant particular acknowledgement.

Dr. J. C. Browne provided the original idea for this research, worked closely with me while the research was in progress, and then displayed laudable patience in the interim between the conclusion of the research and the completion of this Thesis. Dr. Browne was more than an excellent Thesis advisor. Since we first met he has been an encourager, a motivator, a friend and a valued colleague.

Dr. J. L. Peterson provided guidance not only in the development of ideas related to Petri Nets, but also in the difficult task of presenting the research in a concise, understandable form. In every respect, he has been an ideal thesis advisor. His guidance has always been helpful and available; his genuine interest and support have been a constant encouragement.

William "Bill" Berlin, my brother, also deserves special credit for his major role in this Thesis. Bill was particularly helpful in developing the programming design specifications for a TEPN implementation and is the veteran of several very late night sessions required to put the Thesis together.

Behind every thesis there are always those who help with typing, editing, and the other administrative functions necessary for any large project. This Thesis is no exception; indeed, the amount of help that friends so willingly offered as the due date drew near was nothing short of incredible and was a clear demonstration of Christian love and commitment. Altogether, there were twenty people who devoted specific time to the finalization of the Thesis. The eight typists who participated in the project over the last year are all mentioned on the last page of this Thesis (at the bottom of the Vita page), so they will not be listed here. Needless to say, they were most crucial to the success of the effort. They were all efficient and patient and survived many hours of typing and retyping with smiles and enthusiasm. The other twelve people deserving of special recognition helped with the graphics in the Thesis, with editing the drafts, and with delivering the Thesis drafts to my committee members in both Austin, Texas, and Boston, Massachusetts. The graphics were drawn by Jacqui Schultz and Daphne Wilcox. These two were then assisted by Miss Sylvia Payne and Susan Polombo in preparation of the completed figures with labels, markings, etc. Three friends, Blaine Dunn, Donald Gregory and Anne Cowardin, provided final editorial assistance in putting the final document together.

Finally, four individuals, Cam Nelson, Francis Vitegliano, Enrico Barbieri, and Sandra Youla, acted as couriers to insure timely delivery of the documents in Boston and Austin.

Because of my in absentia status, I needed someone in Austin who could handle all of the final administrative details pursuant to graduation and thesis submission. The majority of the credit for work goes to Mrs. Nancy Eatman, Dr. Browne's secretary. Without a doubt, Nancy was the vital link throughout the "thesis generation" process.

Finally, I wish to acknowledge the help of two others who braved the Thesis from beginning to end and put up with me while I was either working or planning to work: Trudi Berlin, my sister, and Cathryn Goff, my close friend. Their support, inspiration, and good humor made an invaluable contribution towards the completion of this Thesis.

Brett Berlin

Arlington, VA
June, 1979

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

This Thesis introduces the Time-Extended Petri Net (TEPN) as a basis for computer system performance modeling, and demonstrates the TEPN's usefulness in modeling a specific computer system problem. We suggest from preliminary modeling results that the TEPN model is extendable to more general application in the performance evaluation of both system and algorithm architectures.

The TEPN model resulted from a project whose goal was to define and implement a modeling system which would:

(1) represent the time-resolved behavior of a set of deterministic interacting parallel processes;

(2) represent the holding of multiple resources; and,

(3) allow specification of models as data structures rather than as programs.[1]

When properly implemented, this system combines much of the power and flexibility of programmed simulation models with the ease of use of queueing models, resulting in a powerful and cost-effective modeling tool.

[1] Queueing models and System Program Graphs are examples of models which are specified as data structures which may then be analyzed by a pre-compiled program which merely "executes" the structure; discrete language simulation models are specified by a computer program and the model itself must be compiled and validated as a computer program as well as a model.

The TEPN is constructed from the general Petri net by

(1) associating type and information content with tokens;

(2) associating with each place a set of functions and state information derived from the tokens at that place; and,

(3) associating input and output templates with each transition to govern the firing of a transition and flow of information through the network.

The deterministic properties of the TEPN structure remain unchanged from those of the general Petri net, if suitable restrictions on the information content of tokens and the range of the functions are made. This combination of Petri net properties and the above extensions results in a model whose power of representation and mode of definition meet the goals stipulated above.

Thesis Organization

Other than the work of Noe and Nutt [Noe and Nutt, 1972 and 1973], the author knows of no major documented research into using the Petri net as the basis for a CPE tool. However, the reader will be helped by a familiarity with the relationship of the TEPN to other major CPE modeling methods, and this background information is presented in Chapter II.

Chapter III supports the TEPN conceptual definition by presenting some basic definitions of the Petri net, upon which the

TEPN model is based and briefly describes other major research concerned with adapting the Petri net as a CPE tool.

Chapters IV and V define and describe the basic concepts and the implementation of the TEPN model, thereby presenting the bulk of the "new" material contained in this Thesis. The reader already familiar with Petri nets and other computer performance evaluation (CPE) modeling methods may want to begin this Thesis with these chapters.

Chapter VI illustrates the usefulness of the TEPN system in analyzing the performance of a complex disk subsystem.

Chapter VII concludes the Thesis with observation concerning the TEPN's future as a CPE tool and suggests some directions for future research.

CHAPTER II

SURVEY OF CPE MODELING TECHNIQUES

This chapter suverys the three major CPE modeling techniques in documented use by the CPE community. We discuss first the discrete simulation methodology, then analytic queueing network models, and finally trace-driven models. Although none of these techniques employ the Petri net, their understanding is fundamental to understanding the objective behind the TEPN network.

Discrete Simulation Modeling

Discrete computer system simulation involves the description of a computer system by a computer probram and the simulation of the interactions within that system as they occur over a discrete time interval. The level of detail of the representation within a discrete simulation model may vary from simulated interactions to a one-to-one mapping of actual system interactions depending upon the computer language used and the problem's requirements.[2]

Most discrete simulation languages were developed in the 1950's and 1960's as tools to study complex processes and system

[2] The kind of problem generally determines the required and/or desired level of model detail, as well as deciding whether the model should be deterministic (i.e., no random variability) or nondeterministic. For example, an analysis of the performance of a new logic circuit would normally require a very detailed model; a logic-circuit level model of a large multiprocessor system would be neither feasible nor useful. Similarly, the logic circuit analysis would probably require a deterministic model, whereas random variability is an important part of most large system models.

design. One of the earliest and best-known simulation languages is IBM's GPSS [Efron and Gorden, 1969]. This language is problem and user centered; it has features which allow the user to describe the flow of work through the processes which are to be simulated in a flowchart-like format. While not used extensively for CPE, the GPSS language is still very popular in many other areas of computer-based simulation.[3]

The most significant special-purpose simulation language for CPE is the Extendable Computer System Simulator (ECSS) [Nielson, 1969]. Developed by RAND Corporation to aid in studies of computer hardware and software systems, ECSS is a superset of the SIMSCRIPT II [Kiviat, et. al., 1969] programming language with several embedded features uniquely required when modeling computer systems. ECSS has been and continues to be applied to large CPE problems, particularly within the Federal Government.

Another type of specialized discrete simulation tool is the packaged simulator, of which the Computer Assisted System Evaluator (CASE) [CASE Manual, 1962] is the most notable example. CASE is a prefabricated model of an arbitrary computer system defined at run time by the user. The input to the package consists of a configuration description and a workload description, while the

[3] CPE uses of GPSS and most other simulation languages (to simulate computer system performance) comprise a relatively small percentage of all simulation applications, as evidenced by the papers contained in the proceedings of the various simulation conferences (proceedings of major simulation conferences are available from the Association for Computing Machinery (ACM) and the Institute of Electronics and Electrical Engineers (IEEE)).

output is a series of reports on the performance of each part of the system as a whole. The package contains intrinsic information concerning the performance characteristics of each piece of "legal" hardware, indexed by make, model number, and vendor. To cover the software impacts upon system performance, CASE not only contains standard factors relevant to the major operating systems it "supports," but also allows the user to specify many parameters which are normally part of the operating system-generation process. Because of the detailed simulation intrinsic to a CASE simulation, the package was used extensively for a number of years. While CASE continues to be useful in many CPE studies, the extreme complexities of systems with a high degree of parallelism have challenged the validity of many results of CASE simulations of such systems.

Discrete simulation modeling's primary advantage is the flexibility of the programming languages available as a medium for building the models. With the proper choice of the host language, one can build a comprehensive simulation model that faithfully represents virtually any system. Furthermore, since the size of a model is only limited by the capability of a machine to handle large programs, this technique may be used to model deterministic systems (or nondeterministic systems) which might be too large to model effectively by other techniques.

The major drawback of the discrete simulation language model is its high cost, both in development and use. The development costs are generally high for two reasons. First, the simulation model itself represents a significant piece of software. Since software development is labor intensive, the costs of simply writing the programs tend to be high. Furthermore, if the systems being simulated are complex then the program logic will also be complex, making program debugging difficult. The second reason for high development costs is the need for careful model verification and validation. Even if the software is completely debugged, it is often difficult to verify that the program faithfully evaluates the model designed for the customer. There is no guarantee that the model as designed is truly representative of the system being studied. Therefore, before the model is used in a production mode, careful validation is necessary to insure useful results. For a large system model, this process can take several weeks, thereby driving costs even higher. The final cost factor is the cost of production use. While the computer resources required to run the model depend upon the model's level of detail and size, even a small, high-level model can require many central processer time units for each unit of simulated time. For example, if an analyst needs to simulate ten minutes of computer system time on the system being modeled and studied, a simulation run may easily require five or six times that in CPU time alone. Furthermore, since many simulation languages require very large run-time systems, even a relatively small model may require a large memory allocation — and correspondingly increased resource costs.

Despite the high costs involved, the capability of building large deterministic models of computer systems is almost unique to the discrete simulation methodology. Thus, simulation models have found wide application in almost every sector of the community of large-scale computer users. ([MacDougall, 1970] is an excellent survey paper discussing computer system simulation in more detail than is appropriate for this Thesis. This paper also contains an extensive annotated bibliography.). However, though used, discrete simulation models are often too expensive for use in studying less expensive systems, a fact which has resulted in the decreasing emphasis of discrete simulation in favor of analytic queueing models, with a corresponding sacrifice in flexibility and determinism.

Analytic Modeling

Within the context of this paper, analytic modeling refers to a technique in which the system being modeled is represented by a mathematical, rather than a simulation, model. In this approach, the analyst seeks to find a mapping to the interactions generic from the system under study to a set of mathematical formulae which can be analytically solved. A large number of techniques fall into the analytic modeling category, including many methods used in operations research, such as linear and integer programming, regression analysis, and queueing network models. Brice [Brice, 1973] discusses mathematical models in detail as they relate to CPE, and the reader is referred to his work for more conceptual background. In this thesis, however, the only analytical technique which will be discussed is the queueing network model.

The queueing network modeling methodology was originally developed within the operations research community as a means of analyzing systems which display certain characteristics of random variability. In 1963, Jackson [Jackson, 1963] discovered a powerful technique for analytically solving queueing network models which met certain criteria. These concepts were independently discovered a few years later by Gordon and Newall [Gordon and Newall, 1967]. Between 1970 and 1976, researchers within the CPE community extended the known concepts of the queueing network model and developed what is now a powerful analytic technique for CPE. As a result of these developments, both the research and industrial communities have been able to develop efficient, user-oriented software packages for automated queueing network analysis, making the queueing network model the most important CPE modeling technique in use today.

The greatest advantage of the queueing network model is its simplicity. The concepts are well-defined and may require minimal training to apply them. User-oriented systems exist which can run on most large computer systems; and models of even large multi-processor systems can be constructed, tested, and validated in a very short time compared to the time required to model the same system using discrete simulation. Also, since the solution is analytic rather than experimental, queueing network models often require less computer resources than other CPE methods might require to obtain comparable results. This inherent simplicity of the queueing model network methodology has made queueing network modeling by far the least expensive modeling method discussed in this thesis.

Despite the utility and simplicity of the queueing network model, the model is limited in three areas: (1) the model is probabilistic in nature and unable to model deterministic processes; (2) the model cannot exactly represent certain types of problems inherent to many parallel processes; and (3) the validity of the model is dependent upon the representation of the data used to generate the probabilistic functions upon which the model is based. The first limitation causes two difficulties. First, there are some problems, such as those dealing with logic design and hardware architecture analysis, which require detailed, deterministic analysis. These problems are not easily solved using queueing networks. Secondly, there are many CPE problems for which it would be desirable to build a model which could be either deterministic or probabilistic, depending upon the data available and the objectives of the analysis.

The second limitation area is caused by several inherent characteristics of the queueing model pointed out by [Browne, _et_ _al_, 1973], who experienced difficulties when modeling a disk input/output subsystem with the disk SEEK:READ/WRITE overlap feature. The difficulties included: (1) queueing networks only approximate the simultaneous holding of multiple resources (such as when both a controller and a disk unit must be held by the same transaction, though not necessarily for the same period of time); (2) queueing networks can only approximate problems encountered by process blocking (such as might happen when a disk seek has been initiated by a controller but is busy when the seek is complete, blocking the original transaction from completion); and (3) queueing networks are limited in capability to model condition-dependent paths in which the conditions are dependent upon more than the location in the network (this is a result of the dependance of much of queueing network theory upon the memoryless property of Markovian systems.

The final area of limitation, that the validity of the model is dependent upon the validity of the probabilistic functions used, is common to all probabilistic systems. Some problems have been resolved by the discovery that some of the common functions are relatively insensitive to inaccuracies in the data used to determine functional parameters (such as the mean of an exponential or negative exponential function). While few researchers would claim that one could depend upon the results of a queueing network model for exact accuracy, many experiments have shown that a well-designed queueing network model can be assumed to be accurate to within ten to fifteen percent, or better. Furthermore, by combining the results of queueing network analysis which such statistical techniques as analysis of variance and confidence intervals, CPE practitioners have shown the queueing network model to be a useful tool for those problems which it is capable of modeling.

Trace-Driven Modeling

Trace-driven modeling is "a technique which combines measurement and simulation for the purpose of evaluating and predicting the performance of systems" [Sherman, 1972]. In particular, a trace-driven model is one which may be driven by either actual workload data or specially-massaged trace data from the operational history of the system under study. For example, a queueing model could be designed as a trace-driven model if, instead of having the workload represented by a stochastic process, the input to the network were fed directly from a system log of actual transaction times. Such a modeling tool has obvious application in a variety of performance analysis problems, particularly in the analysis of system algorithms or hardware configurations.

The most significant work in trace-driven models was that of Anderson [Anderson, 1974], Sherman [Sherman, 1972], and Browne, [Sherman and Browne, 1973; Sherman, Howard and Browne, 1975] between 1969 and 1974 at the University of Texas at Austin. The dissertations of Anderson and Sherman are devoted to the development and use of the trace-driven modeling concepts. Sherman demonstrated the usefulness of trace-driven modeling by building and using a trace-driven FORTRAN simulation model of the University of Texas' UT2 CDC 6600 operating system. Anderson developed a unified trace-driven modeling methodology based upon the System Program Graph (SPG). The SPG is a graph-based representation technique which not only allows the workload to drive the model directly, but also allows the model itself to be represented in terms of a data structure rather than a computer program.

Both Anderson and Sherman demonstrated the primary advantage of the trace-driven technique: the results are not subject to either the smoothing or random behavior effects of using stochastic methods since the inputs to the model are deterministic, rather than stochastic. Since stochastic processes are built to represent "average" behavior, these processes often complicate the investigation of systems in which small pertubations are important to system performance evaluation. The trace-driven model overcomes most of these difficulties by using deterministic data.

A second advantage of trace-driven modeling is that the level of detail of the simulation results may be controlled by varying the detail of the input data, rather than by redesigning and reprogramming the model, which would be required by simulation or queueing network models.

Finally, Sherman observed that the trace-driven model results generally displayed high accuracy which could often be validated by straightforward means. This accuracy resulted in absolute, rather than relative, measures of performance. In other words, analysis of two algorithms could yield the absolute result that algorithm A was, say, ten percent faster than algorithm B (for the data given), rather than the relative result that algorithm A was simply better than algorithm B.

Despite these and other advantages, trace-driven modeling has had only limited practical application. One key reason is that the usefulness of the technique depends completely upon the accuracy of the trace data and the quality of the model that the trace data drives. Accuracy of trace data, of course, is a problem not unique to this methodology, since both discrete simulations and analytic models are dependent upon the same data. The problem tends to be more significant, however, with a trace-driven model because the trace-driven model tends to be more sensitive to minor pertubations in the data, whereas the smoothing effect of stochastic processes tends to eliminate the impact of these pertubations for other methods. The quality of the model that the trace data is driving is not only a function of correct model architecture, but it is primarily a function of the modeling capabilities of the model itself. For example, if the model used is a queueing network model, the trace-driven nature of the approach does not absolve the model from all of the limitations found with queueing network models.

Like the queueing model, the SPG method is limited in its capability to model deterministic processes. Furthermore, the SPG model of a complex system may easily become unwieldy because of the number of nodes included. These and other weaknesses have resulted in very little application of the SPG or other similar trace-driven modeling methods, although Anderson's and Sherman's work have helped to lay an effective groundwork for the development of the TEPN structure.

CHAPTER III

DESCRIPTION OF THE PETRI NET MODEL

The Petri net is an abstract model of information flow first proposed by Carl Petri in Germany, in 1964 [Petri, 1964]. The Petri net was first applied to the study of computer systems by Holt and Commoner [Holt, et.al., 1968; Commoner, et.al., 1971], in 1968, and subsequently enjoyed considerable interest within the Computation Structures Group of Project MAC, from 1968 to 1975. Since 1968, Petri nets have been studied and used both in university and industrial environments to study and design circuits, algorithms, systems, and other processes. In 1977, this research was the subject of a survey paper published in the ACM Computing Surveys [Peterson, 1977], to which the reader is referred for more background information.

Definition of the Petri Net

Figure 3-1 is a graphical representation of a Petri net.

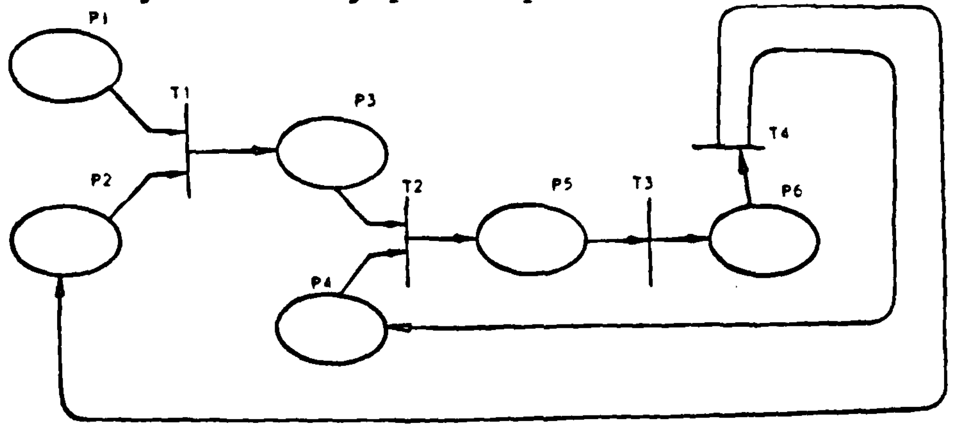


Figure 3-1 Example Petri Net

Informally, the Petri net is a graph structure consisting of two types of nodes, places (pictured by circles) and transitions (pictured by vertical bars), connected by directed arcs. Definition 3-1 formalizes the Petri net concepts pictured in Figure 3-1.

Definition 3-1. Petri Net

A Petri net is defined as a bipartite, directed graph described by the four-tuple, C = (P,T,I,O), where,

$P = \{p_1,\ldots,p_n\}$, a set of places, $n\geq 0$;

$T = \{t_1,\ldots,t_m\}$, a set of transitions, $m\geq 0$;

I is the transition input function, $I{:}T\rightarrow 2^P$

O is the transition output function, $O{:}T\rightarrow 2^P$; and,

sets P and T are disjoint.

In this definition, the connecting arcs are defined by the transition input and output functions, since, for each transition, the input function will yield the set of places connected by arcs directed <u>into</u> the transition, while the output function yields the set of places connected to the transition by arcs directed <u>away from</u> the transition.

Using this definition, the structure of the Petri net of Figure 3-1 would be specified as follows:

PNET=(P,T,I,O), where,

P = {P1,P2,P3,P4,P5,P6}	T = {T1,T2,T3,T4}
I(T1) = {P1,P2}	O(T1) = {P3}
I(T2) = {P3,P4}	O(T2) = {P5}
I(T3) = {P5}	O(T3) = {P6}
I(T4) = {P6}	O(T4) = {P2,P4}

At this point, it is important to understand that the places and transitions of a Petri net are primitive objects, with no associated attributes, functions, or other special meaning.

Marked Petri Net Models

The Petri net structure may be used to represent the structure of information or execution flow in a process, but any study of the dynamic or state properties of a system requires the introduction of another entity, the <u>token</u>.

A token is a dimensionless, uninterpreted object which may "reside" at any place within the network. The number of tokens at a given place is referred to as the <u>marking</u>, or state, of that place, and the vector of markings of all places defines the marking, or state, of the entire network.

Figure 3-2, below, is a marked Petri net, with tokens being represented by dots within the marked places. This marking would be specified by the following vector:

$$M = (0, 1, 0, 1, 0, 0)$$

where,

M_i = the number of tokens currently residing at place i.

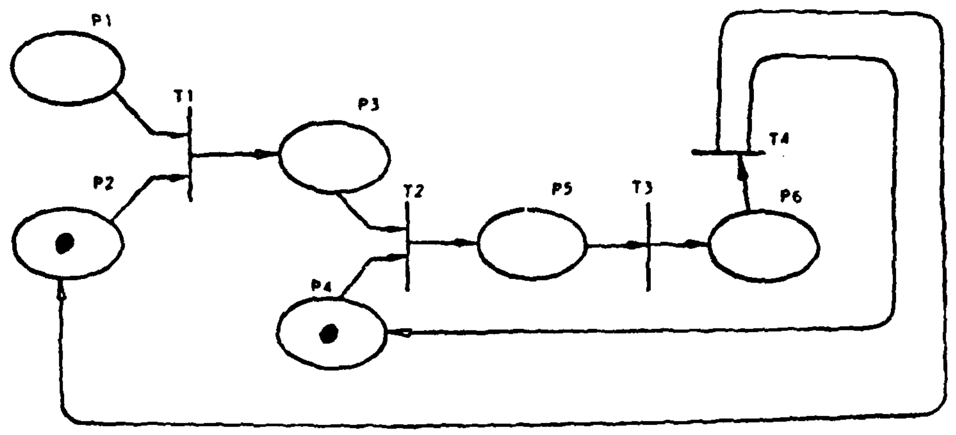


Figure 3-2 Marked Petri Net

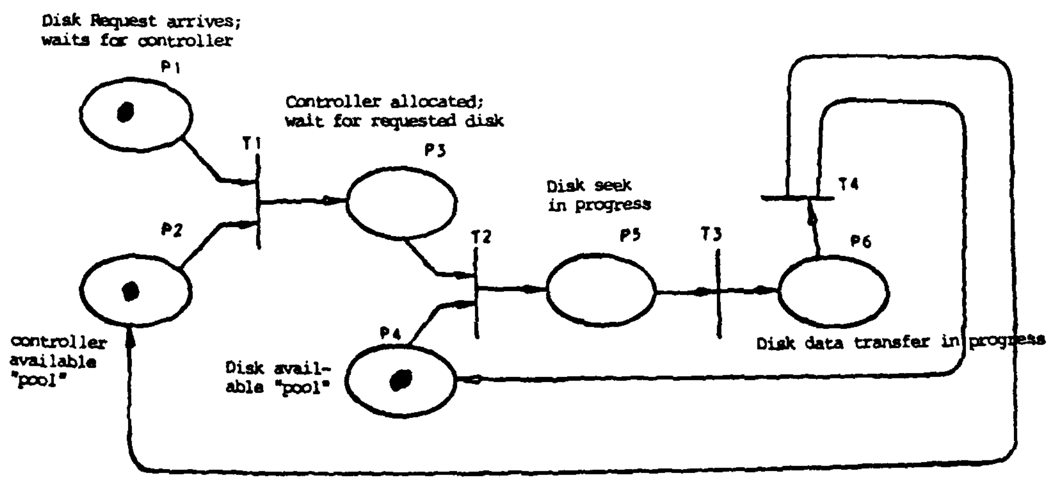

Disk Request arrives; waits for controller
P1
Controller allocated; wait for requested disk
T1
P3
Disk seek in progress
T4
P2
T2
P5
T3
P6
P4
Disk data transfer in progress
controller available "pool"
Disk available "pool"

Figure 3-3 Interpreted Petri Net
With Transition T1 Enabled

Interpreted Petri Net Models

If a Petri net is used to model a specific system, it is necessary to assign a name or interpretation to each node of the network, resulting in an interpreted Petri Net. This interpretation then ascribes meaning not only to the nodes of the network but to the network states, or markings. The net of Figure 3-2 could, for example, be interpreted so as to represent a simple disk subsystem. With this interpretation, shown on Figure 3-3, the marking above might represent the state in which (a) there is one pending disk requests and, (b) both a disk unit and a controller are available.[4]

Execution of the Petri Net Model

When all of the places comprising a transition's input set have a non-zero marking, the transition is said to become enabled, ready to fire, or "happen." If a disk request "arrives" at place P1 (from Figure 3-3), transition T1 will become enabled, and a firing occurs.

A firing occurs deterministically when the specified transition is enabled and involves the following firing rule: one token is removed from each of the input places and a new token is

[4] With this initial marking the Petri net would model a single-controller, single-disk subsystem. A layer subsystem could be modeled by the same structure by adding more controller or disk tokens to the initial marking.

created and placed at each output place. Figure 3-4 illustrates the results firing transition T1 for the Petri net of Figure 3-3, after a new token has been introduced into the net at place P1.

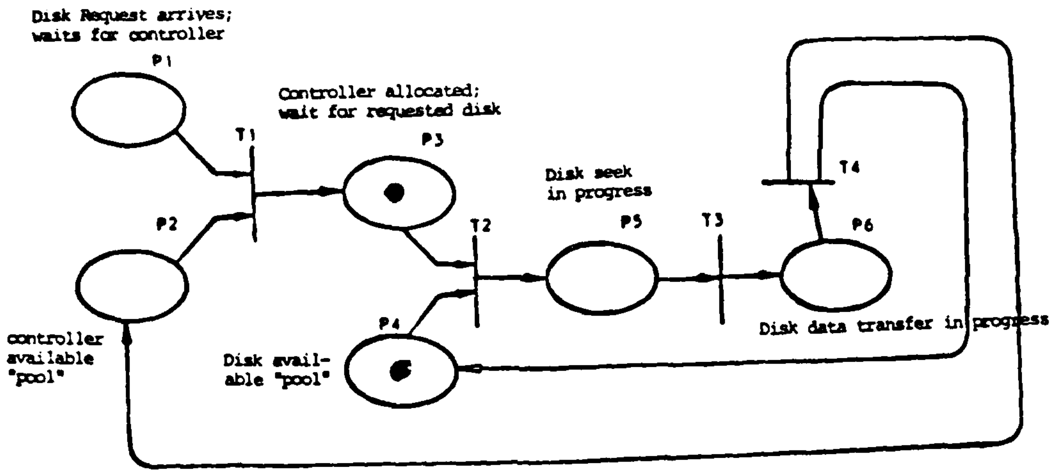

Disk Request arrives; waits for controller
P1
T1
Controller allocated; wait for requested disk
P3
P2
T2
Disk seek in progress
P5
T3
T4
P6
P4
controller available "pool"
Disk available "pool"
Disk data transfer in progress

Figure 3-4 Petri Net After T1 Fires

The Petri Net as a CPE Tool

In 1972, Dr. Jerre Noe [Noe, 1971] of the University of Washington considered the potential of the Petri net as a tool for performance modeling. By building a model of a CDC 6400 operating system and using it to study some basic performance characteristics, Noe illustrated that the Petri net has several attractive properties that make it a potentially tool powerful modeling. The most important of these is the inherently deterministic nature of the model. characteristic gives the model representational accuracy not possessed by stochastic models (such as the queueing network model). A second property is the Petri net's ability to represent a system in varying degrees of detail within the same net. This characteristic affords considerable modeling flexibility without a corresponding decrease in representational accuracy.

Despite its advantages over other CPE tools, the Petri net has yet to become a widely applicable CPE instrument. The primary reason is that the model lacks a crucial attribute required in computer performance analysis: an intrinsic time-resolution mechanism. There is no concept of measurable time for a Petri net execution sequence, and so it is impossible to measure such things as throughput rates and response times. In addition to the time measurement problems, there is no mechanism to naturally represent the flow of specific information across transitions, since all tokens created at firing time are totally independent of any other tokens already in the system. While this is not an insurmountable problem, it does make the modeling of processes much more difficult, particularly when dealing with multiple conditional paths.

The Evaluation Net Model (E-Net.)

Although the Petri Net, per se, did not prove a methodology sufficiently powerful for modeling the performance of computer systems, one of Noe's doctoral students, Gary Nutt [Noe and Nutt, 1972 and 1973], used the Petri net concepts to develop a new structure which would retain positive properties of the Petri net while overcoming some of its limitations. The result was the Evaluation Net, or E-Net. This structure changed several of the basic concepts of the Petri net and added quite a few new features, resulting in a more complex modeling tool. Among the changes included were:

(1) The concept of time was implemented by "delaying" tokens at the transitions. The amount of time was determined by a "transition procedure" and controlled by a global timing mechanism.

(2) In addition to determining firing delays, the transition procedures manipulated tokens and certain global variables.

(3) A set of global "environment variables" was specified and used in conjunction with transition procedures, "resolution procedures," and other functions.

(4) Conflicts between paths were resolved using special resolution procedures activated at transition firing time.

(5) Resolution procedures were also associated with a special class of nodes utilized to aid in resolving ambiguities throughout the network. These nodes contained resolution information defined by the user of the system.

(6) Some tokens were made global. This change, though quite significant, was required by the implementation of time in the transitions instead of the places.

The primary importance of the E-Net is that it was the first major attempt at developing a Petri-net-based modeling system for CPE applications. While the E-Net did not meet with wide acceptance as a usable tool, it did show that Petri net properties could be at least partially preserved, and that a simpler Petri-net-based structure might be useful. The primary drawback of the system was its tremendous complexity, which made even small modeling efforts a major production to all but the true expert. This complexity is contrasted with modeling with queueing networks, which requires a relatively small amount of expertise to construct a reasonable and useful model.

Even with its rather cumbersome complexity, however, Nutt's work represented an effective pioneer effort to the goal of developing and implementing a Petri-net-based CPE model.

CHAPTER IV

THE TIME-EXTENDED PETRI NET: BASIC CONCEPTS

The Time-Extended Petri Net (TEPN) is a modeling structure based upon the Petri net and designed to meet the goals expressed earlier in this Thesis. The advantages of the Petri net, in particular those of inherent parallelism and determinism, have been retained, while the representational capabilities have been extended by implementing places, transitions, and tokens in more elaborate forms.

The Petri net has been extended to allow for the natural representation of time-resolved behavior (changes to the token and place objects) and state-dependent execution paths (changes to the token and transition objects). The TEPN places and transitions are given attributes which allow the direct definition of performance metrics as an inherent characteristic of a model. Finally, the TEPN token is defined as a messenger for activating the places and transitions.

Both the definition and placement of all TEPN attributes are predicated upon a CPE rather than a theoretical view of the Petri net. This view defines an inherent interpretation in which places represent processes and queues, while transitions remain as synchronization primitives that act as state transition arcs. Transitions never represent either processes or queues, nor are they ever part of the state definition of a particular TEPN model. Because of this interpretation, time resolution mechanisms are implemented completely within the place. This includes a place clock, internal place queues, and a token delay time function. In support of the token delay time function within the place, tokens have been extended to allow them to carry external parameters, which can be used by the place token delay time function to compute internal queueing delays. Similarly, transitions have been equipped with special token templates to insure that when tokens are created at transition firing time the tokens will contain values for the parameters required to compute the token delay time within the place.

Implementation of state dependent execution paths requires a token routing mechanism capable of resolving state conditions upon which the token's path is dependent. Since this mechanism affects the order of state transitions, it is implemented in the form of an extension to the Petri net transition called a firing template. Along with a corresponding extension to the Petri net token, the type attribute, the firing template allows for deterministic token routing and provides a mechanism to model the effects of process blocking.

Finally, the token type attribute provides an effective mechanism for resolving the workload of systems into components or activities. While this does not inherently increase the modeling power of the TEPN, it does afford the practical advantage of being able to collapse certain very large Petri net models which do not employ token types into highly compact models with multiple token types. An example of this compaction capability is illustrated in the example TEPN model of chapter six.

Throughout this research, the author has taken great care to retain the modularity and subnetwork independence characteristic of the Petri net. Extensions which create global environments (such as a global synchronizing clock, global tokens, etc.) were avoided, as were modifications which precluded reducibility of the TEPN into a Petri net, in the default case. Despite these efforts the TEPN structure has neither the simplicity nor the elegance of the Petri net. However, it does appear to be a powerful model which remains straightforward enough to be readily usable.

TEPN General Definition

The basic definition of the TEPN model, definition 4-1 below, is structurally equivalent to that of the Petri net, definition 3-1, in chapter three. The Petri net places and transitions are simple primitives with mo dimensions or attributes. Similarly, the token is only a "marker," with no "meaning" of its own. The TEPN structure has the same building blocks but they are no longer simple primitives. Rather, they are potentially complex structures with well-defined attributes. An analogous situation might be found in the structure of chemical molecules. All molecules are built of a basic building block, the atom. For quite a few years, the atom was thought of as a primitive--i.e., it could not be broken down further. Then scientists discovered subatomic particles which are now known to be building blocks for the atom. The new atomic model is able to explain a larger number of phenomena. But, in cases where the subatomic particles do not make any difference, the simpler model is still used and does not compromise the integrity of the more complex structural representation.

In like manner, the TEPN model is able to represent many system interactions which could not be modeled with the Petri net. When the additional modeling power of the TEPN is unnecessary, however, the TEPN model may be treated as equivalent to the simpler Petri net by allowing the more complex TEPN places, transitions, and tokens to default to their simplest form.

Definition 4.1. Time-Extended Petri Net

A Time-Extended Petri Net (TEPN) is a modeling structure defined by the four-tuple, C = (P,T,I,O), where,

$P = \{p_1,\ldots,p_n\}$, a set of places, $n \geq 0$;

$T = \{t_1,\ldots,t_m\}$, a set of transitions, $m \geq 0$;

I is the transition input function, $I: T \rightarrow 2^P$

O is the transition output function, $O: T \rightarrow 2^P$

sets P and T are disjoint.

The next three sections define each of the TEPN building blocks: The TEPN place, transition, and token.

TEPN Place Definition

The state of both the Petri net and the TEPN is determined by the states of the places within the particular net. The transition, on the other hand, is a network element that controls state changes across the net. In this section, I discuss the attributes of the TEPN place. These attributes, all extensions from the primitive place concept found in a Petri net, heavily influence the domain of problems to which the TEPN may be successfully applied, while many of the extensions to the transition affect the solution net's structure.

There are two project goals which led to the major extensions to the Place definition. The first relates to the need for a modeling basis capable of representing time-resolved behavior of networks of processes. The Petri net models <u>structures</u> and structural relationships. With the Petri net, one can study paths of both data flow and control flow through systems of considerable complexity; the Petri net also offers a precise notation for describing the syntax of such system problems. A Petri net cannot, however, describe the time-resolved behavior of a real system. Modeling such problems necessitates that the model incorporate the concept of <u>time</u> and <u>timed performance</u>.

The second project goal is developing a modeling structure which provides a natural, but powerful mechanism for defining and studying various performance metrics within the model itself. Since virtually all performance metrics are defined in terms of the state history of the model, this goal involves the capture of state history information for each of the TEPN places.

The TEPN place conceptually combines the concepts of queues and servers. Functionally, the place receives tokens from input transitions, "stores" the tokens for some elapsed interval, and, when an appropriate output transition is enabled and fires, it "emits" the same token to the output transition. In a CPE model, places might be abstractions of the resources or processes which do not create or destroy tokens; they act as a "way station" for tokens between token creation (at the firing of the input transition) and token destruction (at the firing of the output transition). This definition underscores three characteristics unique to the place. These three are:

(1) Tokens only reside at places, with the result that the state of a net may be defined in terms of the states of places within the net;

(2) The place conserves[5] tokens and could, if so implemented, allow tokens to retain a unique identity; and

(3) While there is no concept of time intrinsic to the Petri net, there is definitely a concept of "waiting," in that a token resides at a place until it is moved by the firing of a transition.

---

[5]Although places conserve tokens -- i.e., tokens are neither lost nor added — this should not be confused with network-wide token conservation as discussed in Petri net literature [Lien, 1976] and [Pererson, 1977].

These characteristics can be combined with a time mechanism to provide a well-formed basis for modeling time-resolved behavior. The TEPN place has been extended from the Petri net place to include an internal clock and other attributes to effectively support such a mechanism. In defining these attributes, we start with the TEPN Place definition.

Definition 4.2 TEPN Place

The TEPN Place is a composite object which receives tokens from adjacent transition nodes and emits the same tokens to adjacent output transition nodes. Each place is internally defined by the following attributes:

(1) Place Clock, a non-negative, asynchronous clock;
(2) Place Active Queue (PAQ), a queue of tokens waiting to become enabled;
(3) Place Enabled Queue (PEQ), a queue of tokens waiting to leave the place;
(4) a set of Place Performance Functions.

The four place attributes represent what the author believes to be the minimal extensions necessary to give the TEPN place the flexibility required for effectively modeling parallel systems. As such, each of these attributes is directly linked to the objectives discussed earlier in this section. The clock provides time-resolution without which most of the performance metrics would lose meaning. The two queues provide a concept of ordering as well as potential for delays (service times) and internal synchronization (including priority schemes). Finally, the performance functions provide a natural, intrinsic mapping from the place state history to the set of integers. The concept of the "state" of a place and the place state history will be discussed further below.

There are several methods for defining each of the four place attributes. For the purpose of this Thesis, it will be sufficient to present the subattributes which form the most primitive characteristics of the clock, queues, and functions mentioned above. In the next chapter we discuss the implementation of the place in terms of the functions which must be programmed in the implementation.

Each of the place attributes is said, for the purposes of this Thesis, to be specified as one of two types of subattributes: configuration attributes and state attributes. Configuration attributes are those which determine the place's internal structure and operational characteristics. If these attributes are not ascribed meaning then the place would be undefined. An example of such an attribute would be the queueing discipline attribute of a queue. A queue cannot be defined or used unless a queueing discipline is associated with it. State attributes, on the other hand, are those which directly affect the place's state when they change as a result of execution of a network and movement of tokens through the place. The current value of the place clock, for example, would be a state attribute.

TEPN Place Clock

This clock is defined as a simple interval clock which is undefined for negative or non-integer values. It is decreasing in that whenever it has a positive value associated with it it automatically and asynchronously decrements by units until the value reaches zero. The clock has no configuration subattributes, and only one state attribute, called TIME.

<u>Definition 4.3 State of a TEPN Place Clock</u>

The state of a TEPN place clock is defined as the current non-negative value of the clock. This value is referred to as the TIME, and is functionally denoted TIME(P), where P references the place whose clock is being observed.

<u>TEPN Place Active Queue and Enabled Queue</u>

A queue is an ordered list defined by an ordering function (queueing discipline) and a function that determines service time. A queue may be further defined to include a maximum size; if it has this attribute and the maximum size is finite, the queue is called a bounded queue. The queues that define each place are queues of tokens which are received by the place during execution of a network. The first queue, the Place Active Queue (PAQ), accepts tokens immediately upon their entry into the place. The service time within the queue is determined by a function called the <u>Token Delay Time Mapping</u> (TDTM), which maps the attributes of the token into the set of integers.

Once the token has left PAQ it enters the Place Enabled Queue (PEQ). This queue has no service time function; the "service time" is determined by the state of the adjacent output transitions. When one of these transitions is ready to accept a token, the PEQ will release a token (if one is available). Although the PEQ does not have an internally defined service time function, it may have a queueing discipline. In this case, certain tokens will be considered to be receiving service while others will be considered to be waiting, although the service time will be indeterminate, since it depends upon the length of time the "enabled" token must wait for an output transition to accept the token.

Definition 4.4 Place Active Queue

The Place Active Queue (PAQ) of place P is the queue of tokens residing within place P which will not be available to leave P prior to a determinate minimum sequence of place clock state transitions. The PAQ configuration is defined by the following attributes:

(1) PAQ queueing discipline,

(2) PAQ Bound, and,

(3) PAQ Token Delay Time Mapping (TDTM).

Definition 4.5 Place Enabled Queue

The Place Enabled Queue (PEQ) of place P is the queue of tokens that are residing within place P and which are not in the PAQ. The configuration of the PEQ is specified by two attributes:

(1) PEQ Queueing Discipline, and,

(2) PEQ Bound.

These two definitions clarify both the functions of and the conceptual differences between the two queues. The first queue deals with tokens waiting because of internal delays caused by the application of the PAQ TDTM; the second, the PEQ, holds the tokens which are delayed due to external factors. It is important to note at this point that the place not only has no control upon these factors, but also has no visibility on them. The expected wait time within the PEQ, therefore, is always indeterminate from the viewpoint of the particular place.

The configuration subattributes of the two place queues are described further below. Except where noted, the corresponding subattributes for both the PAQ and PEQ are conceptually identical, although in implementation they have no _a priori_ relationship.

PAQ and PEQ Queueing Discipline. The ordering function, or queueing discipline associated with each of the place queues may be either a "standard" queueing discipline, such as first-come-first-served (FCFS) or infinite processor (IP), or it may be a specially defined function. Regardless of how it works, however, the queueing discipline determines both how many tokens may be "active" or "enabled" at any given time, and the degree of dependence or independence of tokens passing through the same place. One result of this is that the queues provide a natural partitioning of the place's resident tokens into four sets, or states. For purposes of this Thesis, these four states shall be defined as follows:

(1) WAIT: token in PAQ, not being served

(2) ACTIVE: token in PAQ, being served

(3) ENABLED-BLOCKED: token in PEQ, not available to leave place

(4) ENABLED-READY: token in PEQ, available to leave place.

Queue Bound. Each of the queues may be bounded by some finite value. This value will determine the maximum size to which the entire queue may ever grow. The queues may be separately or jointly bounded. A joint bound treats the entire place as a large queue by stating that only a limited number of tokens may reside within a place at any given time. If a place is only assigned a joint bound then there is no restriction upon how the tokens will be distributed among the two queues. In addition to or in place of an explicit joint bound, each queue may be separately bounded. If there is no explicit joint bound but both queues are bounded then the place is said to have an implicit bound equal to the sum of the individual queue bounds. If there is an implicit or explicit joint bound then the place is said to be bounded, as expressed by Definition 4.6 below.

<u>Definition 4.6 Bounded Place</u>

A place, P, is said to be <u>n-bounded</u> if the PAQ and PEQ of P are jointly bounded by n. If n is the smallest integer such that P is still n-bounded, then n is said to be the minimal bound of P.

The concept of a finitely bounded place allows modeling of systems which have severe restrictions upon the "processing" capability of specific nodes, or which display performance characteristics very sensitive to increased "workload." However, it should be noted that while no real systems are in fact infinite servers there are systems where it is useful to determine the "natural" steady state conditions that would exist if limits did not exist. Therefore, there are many cases in which the places would be unbounded.

<u>Token Delay Time Mapping (TDTM)</u>. Each Place Active Queue has a Token Delay Time Mapping (TDTM) associated with it. This attribute defines a mapping from the domain of tokens into the range of non-negative integers, and determines the minimum internal time delay required before the token may become enabled and enter the Place Enabled Queue. The passing of this time delay is indicated by commensurate changes in the state of the place clock. This delay does not include any wait time experienced prior to the application

of the function to the token. For example, if the queueing discipline is FCFS (first-come-first-served), then tokens would have to wait in line until the PAQ "server" was ready. Then the delay TDTM would be applied to the token and the token would reside within the "server" for exactly the amount of delay resulting from the application of the token function to the token, after which it would be released to the PEQ.

Definition 4.7 Place Token Delay Time Mapping

The Place Token Delay Time Mapping (TDTM) is a transformation which maps the domain of tokens into the range of non-negative integers.

The State of the PAQ and PEQ. In addition to its configuration subattributes, each queue has an implicit set of state subattributes which define the state of the queue and help to define the state of the place. Definitions 4.8 and 4.9 define these state attributes.

Definition 4.8 State of the Place Active Queue

The state of the place active queue is defined by the following state attributes:

(1) The sequence of tokens within the queue in the WAIT state, and

(2) The sequence of tokens within the queue in the ACTIVE state.

Definition 4.9 State of the Place Enabled Queue

The state of the place enabled queue is defined by the following state attributes:

(1) The sequence of tokens in the ENABLED-BLOCKED state, and

(2) The sequence of tokens in the ENABLED-READY state.

TEPN Place Performance Functions

The TEPN Place extensions discussed above provide for a well-defined and flexible ability to model time-resolved behavior without altering the basic determinism and subnet independence of the Petri net. The final extension to the Petri net Place is a mechanism to introduce the definition of performance metrics into the modeling structure itself. This aspect of the TEPN is somewhat unique from most modeling methods in that it merges the modeling tools with the data analysis tools, thus simplifying the CPE modeling process. This mechanism also aids the analyst in validating the model by providing a reliable link between the model and the model's apparent results. In other words, by allowing for predefined, well-tested internal performance metrics, the TEPN structure definition may eliminate many of the problems encountered when the output has not been sufficiently validated to assure agreement with internal results.

Definition 4.10 Place Performance Function

A Place Performance Function is a mapping from the sequence of execution states of the place over a specified time interval into the set of real numbers.

Another way to state definition 4.10 is to say that a performance function is a mapping from the results of the execution of a TEPN model into a real number which represents some standard (or, perhaps, non-standard) performance metric such as throughput, average wait time, etc. These metrics are all results of some "summarization" operation taking into account every state that the place has experienced during the specified time interval.

Definition 4.11 TEPN Place Execution State

The TEPN Place Execution State is a function of time and is defined as a vector containing the values of all state subattributes of the Place Clock, the Place Active Queue, and the Place Enabled Queue.

By combining Definitions 4.10 and 4.11, above, we see that the domain of each place performance function is a sequence of vectors containing the values of state subattributes over a specified time interval. For a given model over a given time interval this sequence is the model's state history.

<u>Definition 4.12 State History</u>

The state history of a TEPN place is the sequence of execution state vector values over a specified time interval. The state history for place P over the interval (t1, t2) is designated $S_p(t1,t2)$.

We may now give an alternate definition for a performance function.

<u>Definition 4.13 Place Performance Function (alternate definition)</u>

A Place Performance Function is a mapping of the place state history into the set of real numbers. In functional notation,

$$PPF = f(S_p(t1,t2))$$

where (t1,t2) is the time interval over which the performance is being measured.

One should note that at least for the conceptual definition the internal specifications of any particular performance function are to be avoided. Part of the significance of this feature in the TEPN definition lies in the fact that the functions are defined as part of the construction of each model.

TEPN Token Definition

Like the TEPN places, the TEPN token is a composite data object. Definition 4.14 defines the TEPN token; the two token attributes are then discussed in more detail below.

Definition 4.14 TEPN Token

The TEPN token is a composite object defined by the following attributes:

(1) the token type; and

(2) the token functional attribute set.

Token Type

The objective of the token type is to provide the token with an identifier that can be used to control the token's execution flow path. This attribute is used in conjunction with the transition firing template and is determined at token generation time (during the firing of a transition).

Token Functional Attribute Set (FAS)

At token generation time, each token is built to include a set of intrinsic functions which return integer or boolean values. These functions return values which are used by the place token delay time function to determine the "service time" required of the token upon arrival at the Place Active Queue (PAQ). Within the TEPN's conceptual definition the exact nature of these functions are not defined, since they are arbitrary in nature and are uniquely defined for each token created and are incorporated within the transition's token template. Definition 4.15 does, however, describe the concept in more formal terms.

Definition 4.15 Token Function Attribute Set (FAS)

The token function attribute set is a set of single-valued, integer or boolean functions of the form (FNAME, value), where "FNAME" is the function name and "value" is the current functional integer or boolean value.

Marked TEPN Models

A TEPN is said to be marked when one or more of the TEPN places belonging to the network have one or more resident tokens. Similar to the Petri net, the marking of a TEPN determines most of the state attributes of a given network. Unlike the Petri net, however, in which the marking is simply a count of the number of tokens at each place, the TEPN marking may be expressed not only in terms of token counts, but also in terms of specific tokens and their current attributes. Because of this potential for added complexity, this Thesis defines two markings, the internal marking and the external marking. As their names might suggest, the criteria used in developing the definitions of each of these marking concepts was the "view" of the internal versus external "observers" of each place and of the network as a whole.

Definition 4.16 defines the external marking of a TEPN. This definition is equivalent to the definition of a Petri net marking for a Petri net that contains typed tokens[6]. Although the information required to derive the external marking exists as a subset of any more comprehensive inventory of the TEPN's resident tokens, a separately defined external marking simplifies situations where more complex information would not be required. Furthermore, it is important to recognize that the external marking is the marking which could be observed via a "global" view, since no place or token (or transition) attributes are global by definition.

Definition 4.16 TEPN External Marking

The External Marking of a given TEPN, N, is a K-vector, where k is the number of places in the TEPN, and the ith element of the vector is an m-vector in which m is the integer count of token types currently defined within place P_i, a unique place within TEPN N, and the jth element of the vector is the count of P_i resident tokens of token type TYj.

The internal marking is described by the sets of actual tokens residing at each place. This is the view from within the

[6] Peterson [Peterson, 1979] and others discuss Petri nets with "colored", or typed tokens, as a manner of simplifying Petri nets. As Peterson points out, any Petri net with typed tokens, of a finite number of types, can be shown theretically equivalent to some Petri net without typed tokens. This equivalence does not necessarily hold up within the TEPN environment, except in the default case (in which the TEPN becomes a Petri net).

place itself. The internal marking includes not only the number and types of tokens within each place, but the current state of each token and, in some cases, the function attribute values for specific tokens within specified places. Definition 4.17 defines this marking concept. Note that the marking is characterized by sets, and not token sequences or queues. The ordering of tokens within each place queue is not a factor in the marking although this ordering is important to the place state.

Definition 4.17 TEPN Internal Marking

The Internal Marking of a given TEPN, N, is a (1 x k) vector, where k is the number of places in the TEPN, and the ith element of the vector is a partition of the set of tokens residing within place P_i, of TEPN N, expressed as an ordered four-tuple,

$M_i = (TS_w, TS_a, TS_{eb}, TS_{er})$, where,

TS_w = set of place tokens in WAIT state,

TS_a = set of place tokens in ACTIVE state,

TS_{eb} = set of place tokens in ENABLED-BLOCKED state, and

TS_{er} = set of place tokens in ENABLED-READY state.

M_i is called the Local Internal Marking for place P_i.

TEPN Transition Definition

Like its Petri net counterpart, the TEPN transition is a synchronization primitive which controls the flow of tokens across a TEPN model. Similar to the Petri net transition (and unlike the

TEPN place), TEPN transitions neither hold tokens nor record the passage of time. All actions which happen at a transition are considered to happen instantaneously; in this sense, the transition can be likened to a switch required to control local state changes. Since there is no concept of action or token storage at the transition, the transition has no attributes which impact the state of its parent net--all transition attributes are configuration attributes whose values are determined as part of the model definition.

The TEPN transition is formed from the Petri net transition by adding two attributes: one impacts the enabling process, the other controls the creation of new tokens during the transition firing process[7]. The first attribute is the transition firing template. The firing template controls the transition enabling process by selectively filtering input tokens and allowing only tokens of predetermined types to enable the transition.

The second attribute is called the transition token template set. Since TEPN tokens are defined with attributes there must be a mechanism for building new tokens with the attributes requisite to correct operation of the model.

[7] The TEPN retains the basic transition firing and enabling process of the Petri net, as discussed in Chapter 3. While the transition firing template and token templates alter some of the specific conditions under which transitions may become enabled and create new tokens, the basic concepts of enabling and the firing remain intact.

Definition 4.18 defines the TEPN transition; each of the two attributes are then described and formally defined following the general transition definition.

<u>Definition 4.18 TEPN Transition</u>

The TEPN transition is a composite object which receives tokens from adjacent input place nodes, creates new tokens for transmission to the output places, destroys the input tokens, and transmits the new tokens to the output places for which they were created. The configuration of each transition is defined by the following two attributes:

(1) TFTS, a set of transition firing templates; and,

(2) TTTS, a set of transition token templates

<u>TEPN Transition Firing Template</u>

In a Petri net, a transition is said to be "enabled" whenever there is a token residing at each of its, input places. Extending the TEPN place to include an internal token delay mechanism changes the definition of an enabled transition. Rather than simply requiring that each input place have a token, a TEPN transition may only be enabled by tokens that have been "released" or "enabled" by the place after the required internal wait time. The transition firing template extends the transition enabling conditions a step further by allowing selective enabling based upon the types of tokens residing at the transition input places. More specifically, a transition firing template acts as a filter that only allows tokens of prespecified types to enable the transition. When there are two or more input places, the firing template specifies not only the token types that may enable a given transition, but the allowable combinations of input token types as well. For example, if there are three input places which may emit tokens of types A, B, and C, the enabling patterns might be (A,A,A), (B,B,B), and (C,C,C). In this case, the transition will fire only when there are tokens of the same type at all input places. Even though all three types are "legal", the transition would not be enabled by any combination not specified above (e.g., (A,A,B), (A,B,C), etc.).

The primary need for this extension arises from the difficulty in the Petri net of modeling the deterministic performance of systems with dynamically chosen multiple execution paths. For example, when modeling the performance of a disk subsystem, the modeling objective may require detailed information concerning the performance of the system on a channel by channel and disk by disk basis. If the workload can be shown to be evenly spread across all of the subsystem components, this objective does not present serious problems, since the aggregate average performance will reliably model the performance of individual components. However, in cases where certain channels, controllers, and disk drives are more heavily utilized, such as a system in which the system software storage units may be utilized with several times the frequency of use of most other units, the model must allow for transactions (possibly represented by tokens in this case) to deterministically specify the execution path, a feature not intrinsic to the Petri net model. In a Petri net, when there are multiple paths emanating from a place there is no way of controlling which path will be taken. The TEPN transition firing template solves this dilemma by allowing only prespecified token types to enable the transition.

In addition to the extended modeling power that it affords the TEPN model, the firing template often reduces the number of nodes required to model systems by allowing several independent paths originating in a common place node to be collapsed into one path with multiple firing templates. This advantage is demonstrated by the model presented in Chapter VI.

Definition 4.19 TEPN Transition Firing Template

A firing template of transition T is a function from the set of local internal markings of all places in I(T), the set of input places to transition T, into the boolean set, ENABLED, NOT ENABLED . In functional terms,

$TFT:LIM_i \rightarrow \{ENABLED, NOT\ ENABLED\}$,

where, LIM_i is the current local internal marking of place P_j, the jth member of I(T).

TEPN Transition Token Template

Each transition's output token template set defines the tokens to be created by that transition during a firing sequence. Because TEPN tokens have types and attributes, and because these attributes are not global, each transition must have the information necessary to build tokens with the "proper" attributes. The mechanism for doing this is the token template, which, when executed, uses the transition's input tokens as inputs and generates a single token as the output. The set including one token template for each output place is known as the token template set.

Conceptually, an output token template is a function from possible sets of input tokens possible output tokens.

There are a few attributes of the token template which should be noted. First of all, while the input token set is used as input to the token templates, the token templates are completely independent of the transition firing templates. Secondly, the concept of the token template is independent of the particular attributes of the tokens. The exact attributes, etc., are dependent upon the system being modeled. Definition 4.20, below, formally defines the transition output token template.

Definition 4.20 Transition Token Template

A Token Template is a function from the set of sets of possible input tokens into the set of possible output tokens. More specifically, the token template defines the transformation from the set of input tokens for a given transition to a single token to be created during the transition's firing sequence and sent to an associated output place. In mathematical notation:

$\mathrm{TOKEN}_i = \mathrm{TTT}_i\ (\mathrm{TIS})$

where,

TOKEN_j is the token being sent to place P_j,

TTT_j is the token template associated with place P_j,

TIS is the sequence of tokens which enabled the transition (one from each member of I(T)),

P_j is the $\underline{j\text{th}}$ place in the set O(T), $1 \leq j \leq |O(T)|$.

Execution of the TEPN Model

Just as the TEPN marking was defined in terms of two observers, so the model execution might be described.

The external view of TEPN execution is generally identical to that for the Petri net. When there is an eligible token at each transition input place the transition "fires" by removing the enabling tokens from the input places and creating new tokens for the output places.

The internal view of the execution introduces both the firing and token templates within the transition and the time delay mechanisms within the place. Each of these extensions from the Petri net have been individually presented earlier, and will not be discussed again in this section. It is important to recognize that the introduction of these features makes both net performance analysis and TEPN implementation as CPE tools more complex tasks.

CHAPTER V

THE TIME-EXTENDED PETRI NET: IMPLEMENTATION OF THE TEPN MODELING SYSTEM

This chapter presents machine independent module and function specifications which outline an implementation of the TEPN as a CPE tool.

The format of these specifications is a data-object centered decomposition using the methodology proposed by D. L. Parnas Parnas, 1972 in his extensive work on the specification of software systems. This is a conceptual design stage and is transportable to many potential host system implementations.

The first section of this chapter presents the objectives of the TEPN modeling system and relates the system to the overall goals of this Thesis. The second section overviews the TEPN system design, concepts and organization. The third and final section is a set of highest level module specifications.

Objectives of the TEPN Modeling System

The implementation of the TEPN has two major goals. The first, and possibly most important goal is to provide an effective test bed for evaluating TEPN models of non-trivial systems. Because of computational complexity, it is virtually essential to use automation to execute a TEPN model since at this time there are no known techniques for analytically "solving" a TEPN model, as there are for solving queueing models.

The second major objective of the TEPN system implementation is to evaluate the TEPN's cost-effectiveness as a modeling tool. There are several areas which make up this trait, including,

(a) the time to build a model,

(b) the difficulty of validating models,

(c) the efficiency of the actual model execution, and

(d) the usefulness of the output.

System Design Overview

The TEPN Modeling System consists of two subsystems. The first, subsystem DEFINE.TPN, provides the mechanism for the user to define the structure, semantic, and performance evaluation attributes of a TEPN model. The second subsystem, EXEC.TPN, initializes and executes a TEPN model according to user specifications and outputs the performance statistics resulting from the model's execution.

Each of these subsystems is decomposed into Parnas modules centered upon the data structure and other major internal design decisions.

Subsystem DEFINE.TPN

The purpose of this subsystem is to assist the user in designing accurate TEPN models and translating these designs into internal structures that can be efficiently used to execute the models. Figure 5-1 illustrates these objectives in terms of inputs and outputs to the subsystem as a whole. The inputs, including the TEPN structure, semantic attributes, and performance characteristics, result from the manual network design process and are input in the TEPN Network Definition Language (NDL). The outputs are a network description in another format to assist the user in ongoing model development and the TEPN description file for the TEPN execution subsystem.

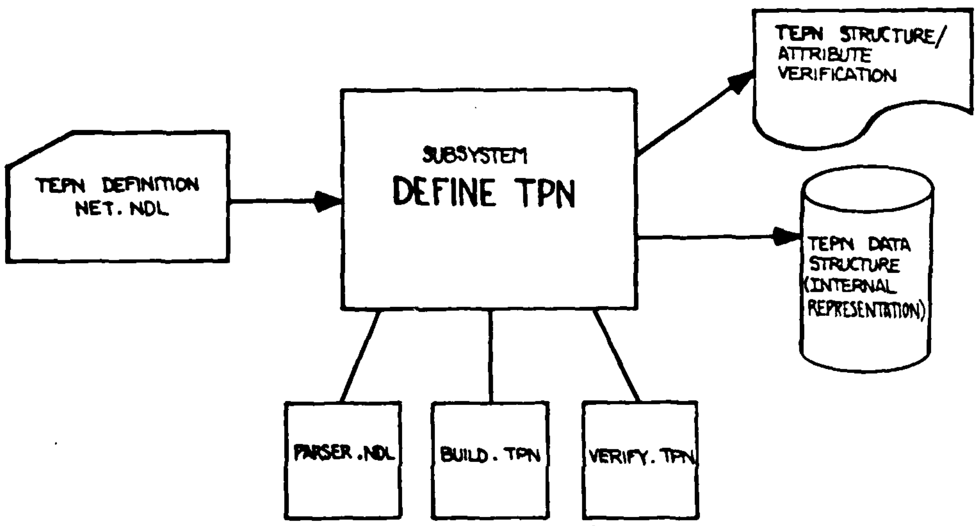

TEPN STRUCTURE/ ATTRIBUTE VERIFICATION
SUBSYSTEM DEFINE TPN
TEPN DEFINITION NET. NDL
TEPN DATA STRUCTURE (INTERNAL REPRESENTATION)
PARSER.NDL
BUILD.TPN
VERIFY.TPN

Figure 5-1 Subsystem DEFINE.TPN

DEFINE.TPN consists of three modules:

(1) <u>MODULE PARSER.NDL</u>

Each DEFINE.TPN system is partially characterized by a user-interface language called the Net Definition Language(NDL). The NDL Parser accepts NDL strings and translates them into a TEPN model that can be used by the execution subsystem.

(2) <u>MODULE BUILD.TPN</u>

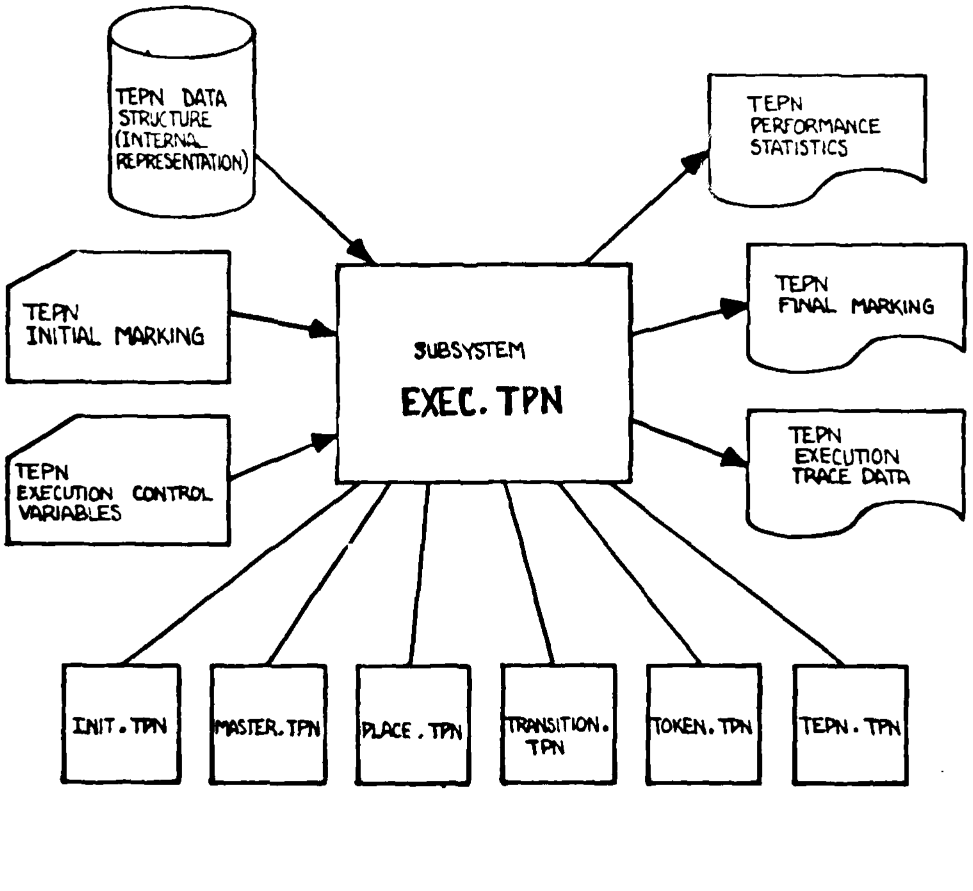
This module consists of a set of functions which may be used to build the internal structure of the TEPN model as required by the execution system. This module shares data structure with TEPN, PLACE, and TRANSITION modules in the EXEC.TPN subsystem, since the functions must have knowledge of the internal structure of these TEPN building blocks within EXEC.TPN. For this reason, BUILD.TPN is itself partitioned into submodules, each of which only deals with a single data structure (i.e., only the PLACE, or the TRANSITION — not both).

(3) <u>MODULE VERIFY.TPN</u>

This module may be invoked by the user as a means of verifying that the final TEPN resulting from the original NDL definition is what it was intended to be. The functions in this module traverse the data structure and output a user-oriented description of the TEPN structure and attributes as they are currently defined.

Subsystem EXEC.NET

The EXEC.NET subsystem handles all of the work involved in executing and observing the performance of a TEPN network. Figure 5-2 schematically illustrates the work performed by the subsystem.


TEPN DATA STRUCTURE (INTERNAL REPRESENTATION)
TEPN INITIAL MARKING
TEPN EXECUTION CONTROL VARIABLES
SUBSYSTEM
EXEC. TPN
TEPN PERFORMANCE STATISTICS
TEPN FINAL MARKING
TEPN EXECUTION TRACE DATA
INIT.TPN
MASTER.TPN
PLACE.TPN
TRANSITION.TPN
TOKEN.TPN
TEPN.TPN

The subsystem consists of six modules, each of which are described below:

(1) MODULE INIT.TPN

This module accepts an unmarked TEPN (probably from Module BUILD.TPN) and applies an initial marking to it, thus beginning the execution cycle. This module is also responsible for initializing and operating any "token generators" required to simulate workload throughput. The designs of the initial marking format and the token generation mechanisms are localized within this module.

(2) MODULE MASTER.TPN

This module keeps track of upcoming events in the execution of models. This module maintains an event timing (and "alarm") system which allows for orderly system execution.

(3) MODULE PLACE.TPN

The place data structure design is embedded within this module, which consists of all functions used to manipulate the place structure during model execution. In addition to the functions required to execute the place, this module also includes the set of functions required to support the BUILD.TPN module of subsystem DEFINE.NET (although for purposes of efficiency, these build routines are embedded directly in the BUILD.TPN module.

(4) <u>MODULE TRANSITION.TPN</u>

Similar to Module PLACE.TPN, the TRANSITION.TPN module defines the design of the trainsition data structure and supports all functions required to execute the TEPN transition, as well as those required to build and initialize the structures.

(5) <u>MODULE TOKEN.TPN</u>

The token is dynamically created and destroyed during network execution and this module localizes all functions related to those processes. In particular, this module consists of the functions which build tokens from transition token templates as well as from the initial marking defined by Module INIT.TPN.

(6) <u>MODULE TEPN.TPN</u>

This module maintains the internal data structure that views the TEPN model as a whole, rather than in terms of individual places or transitions. In terms of the original TEPN definition (definition 4.1, of Chapter IV), this module maintains the TEPN Input and Output functions, $I(T_i)$ and $O(T_i)$, respectively.

TEPN System Specifications

The remainder of this chapter presents the programming Design specifications for the TEPN Modeling System. Three of the modules, Modules TOKEN.TPN, TRANSITION.TPN, and PLACE.TPN, form what could be considered the "heart" of the system since they are the direct implementation of the definitions presented in Chapter four. The reader will notice that these modules' specifications closely follow their formal definitions in Chapter IV.

In studying these specifications, it is important that the reader understand their purpose. These are not computer programs. Nor, for that matter, do they define specific algorithms or data structure designs. Rather, they are designed to communicate the functions and operations that are part of each module and information required to interface with each module.

The format for the specifications is uniform throughout. For each module, there is a "header section" consisting of the module name and a general description of the modules inputs and outputs. Each header section is then followed by a list of each function defined for that module. In the case of the basic TEPN data objects, these functions include not only actions upon the object but the object's attributes as well. The format for the function specifications is:

<u>Function</u>: the function name with parameters in parenthesis

<u>possible values</u>: if the function is an attribute which itself takes values, the value range is specified here

<u>parameters</u>: the names and data types of any input or output parameters

<u>effect</u>: the external effect that the function will have upon parameters and other functions (including calls to other functions)

<u>initial value</u>: if the function can take on a value (i.e., if the "possible values" attribute is other than "none"), this attribute specifies what, if any, initial value is assumed.

Subsystem DEFINE.TPN

Module Implementation Specifications

Module: PARSER.NDL

Module Description: Accepts a TEPN defined using the TEPN Network Definition Language (NDL) and translates the NDL strings into calls to functions of Module BUILD.TPN. These function calls are grouped into a network definition "meta file" which may either be saved or immediately passed to the BUILD.TPN module for processing.

Impact upon other modules: Prepares calls to BUILD.TPN

Data Structure Unique to Module: NDL String

Function Description: This module is described by a single, general function that parses NDL strings and creates equivalent function calls.

Function: PARSE.NDL (NET.NDL, NET.BLD)

possible values: none

parameters: file NET.NDL; NET.BLD

effect: none

<u>Module</u>: PARSER.NDL

<u>Module Description</u>: Accepts a TEPN defined using the TEPN Network Definition Language (NDL) and translates the NDL strings into calls to functions of Module BUILD.TPN. These function calls are grouped into a network definition "meta-file" which may either be saved or immediately passed to the BUILD.TPN module for processing.

<u>Impact upon other modules</u>: Prepares calls to BUILD.TPN

<u>Data Structure Unique to Module</u>: NDL String

<u>Function Description</u>: This module is described by a single, general function that parses NDL strings and creates equivalent function calls.

Module: BUILD.TPN

Module Description: Accepts a set of function commands prepared by PARSER.NDL and builds the internal TEPN structure which is then passed on to subsystem EXEC.TPN system. May also operate using direct command file (not prepared by PARSER.NDL).

Inpact upon other modules: none

Data Structure Unique the Module: TEPN Data Structure (PLACE, TRANSITION, TEPN)

Function Description: These functions form "virtual" submodules, each of which centers upon a specific data structure and is conceptually "tied" to an appropriate module in subsystem EXEC.NET. For example, each place is built using functions (e.e., BLD.PLA or NEW.PLA) within BUILD.TPN but specially designated with a .PLA suffix. These functions whare "knowledge" of the internal place data structure with module PLACE.TPN of subsystem EXEC.NET.

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: DEFINE.TPN MODULE: PLA

```
FUNCTION:  NEW.PLA (PNAME)
        POSSIBLE VALUES:  PLACE-ID
        PARAMETERS:  ANUM-STRING PNAME
        EFFECT:  NONE
        INITIAL VALUE:  NONE
```

DESCRIPTION: Each place is assigned a general name by the user and an internal name by the system. Therefore, one of the first functions of creating a new place is to assign

```
FUNCTION:  BLD.PLA (P,TYPE,TIN,TOUT)
        POSSIBLE VALUES:  NONE
        PARAMETERS:  PLACE-ID P; PLACE-TYPE TYPE; TRASET TIN,TOUT
        EFFECT:  CLOCK.TYPE(P) = TYPE;
                 CALL ADDPLA.NET(P,TIN,TOUT)
```

DESCRIPTION: Builds the place internal structure (without parameters). The type ('default' or 'active') determines the type of structure that will be used.

```
FUNCTION:  BLDPAQ.PLA (P,QDISC,TDTM,BND)
        POSSIBLE VALUES:  NONE
        PARAMETERS:  PLACE-ID P; QDISCIPLINE QDISC;
                     INTEGER FUNCTION TDTM; INTEGER BND
        EFFECT:  QDISC.PAQ(P) = QDISC;
                 TDTM.PAQ(P) = TDTM;
                 BOUND.PAQ(P) = BND;
```

DESCRIPTION: Initializes the attributes of the Place Active Queue.

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: DEFINE.TPN MODULE: PLA

```
FUNCTION:  BLDPEQ.PLA (P,QDISC,BND)
        POSSIBLE VALUES:  NONE
        PARAMETERS:  PLACE-ID P; QDISCIPLINE QDISC; INTEGER BND
        EFFECT:  QDISC.PEQ (P) = QDISC;
                 BOUND.PEQ(P) = BND;
```

DESCRIPTION: INITIALIZES THE ATTRIBUTES OF THE PLACE ENABLED QUEUE.

```
FUNCTION:  BLDPPF.PLA (P, PPFSET)
        POSSIBLE VALUES:  NONE
        PARAMETERS:  PLACE-ID P; PPFSET PPFSET
        EFFECT:  PPFS (P) = PPFS (P) + PPFSET
```

DESCRIPTION: THE ACTUAL PLACE PERFORMANCE FUNCTION DEFINITIONS ARE BOTH IMPLEMENTATION AND USER DEPENDENT. THIS FUNCTION ASSUMES THAT THE PPF NAMES IN PPFSET HAVE BEEN DEFINED EITHER BY THE IMPLEMENTATION SYSTEM (SUGGESTED FOR MOST COMMON FUNCTIONS) OR THE USER.

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: DEFINE.TPN MODULE: BUILD

```
        FUNCTION:  NEW.NET (NETNAME)
                POSSIBLE VALUES:  TEPN-ID
                PARAMETERS:  ANUM-STRING NETNAME
                EFFECT:  NONE
                INITIAL VALUE:  NONE

DESCRIPTION:  ASSIGNS INTERNAL SYSTEM NAME TO USER-GIVEN NAME FOR
THE ENTIRE TEPN MODEL.


        FUNCTION:  ADDPLA.NET (P, TIN, TOUT)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  PLACE-ID P; TRASET TIN, TOUT
                EFFECT:
                 FOR EACH TRANSITION T IN TOUT, TINSET(T)=TINSET(T)+P
                 FOR EACH TRANSITION T IN TIN, TOUTSET(T)=TOUTSET(T)+P;

DESCRIPTION:  ESTABLISHES ARCS FROM PLACE P TO ITS OUTPUT TRANSITIONS
AND FROM ITS INPUT TRANSITIONS TO P.)


        FUNCTION:  ADDTRA.NET (T, PIN, POUT)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  TRANSITION-ID T; PLASET PIN, POUT
                EFFECT:  FOR EACH PLACE P IN PIN, TINSET(T)=TINSET(T)+P
                         FOR EACH PLACE P IN POUT, TOUTSET(T)=TOUTSET(T)+

DESCRIPTION:  ESTABLISHES ARCS BETWEEN TRANSITION T AND ITS INPUT
AND OUTPUT PLACES.
```

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: DEFINE.TPN MODULE: BUILD

```
        FUNCTION:  NEW.TRA (TNAME)
                POSSIBLE VALUES:  TRANSITION-ID
                PARAMETERS:  ANUM-STRING TNAME
                EFFECT:  NONE
                INITIAL VALUE:  NONE

DESCRIPTION:  ASSIGNS AN INTERNAL REFERENCE NAME TO THE USER-GIVEN
TRANSITION NAME.


        FUNCTION:  BLD.TRA (T,PIN, POUT)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  TRANSITION-ID T; PLASET PIN, POUT;
                EFFECT:  CALL ADDTRA.NET (T,PIN,POUT)

DESCRIPTION:  BUILDS THE INITIAL INTERNAL STRUCTURE FOR TRANSITION T
AND CALLS FUNCTION TO INSERT TRANSITION INTO THE NET STRUCTURE.D


        FUNCTION:  BLDTT.TRA (T,TT)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  TRANSITION-ID T; TOKTEMP-ID TT;
                EFFECT:  TTS(T) = TTS(T) + TT

DESCRIPTION:  BUILDS THE TOKEN TEMPLATE AS ATTRIBUTE OF TRANSITION T.
```

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: DEFINE.TPN MODULE: BUILD

```
        FUNCTION:  BLDFT.TPA (T,TFT)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  TRANSITION-ID T; FIRETEMP-ID TFT
                EFFECT:  TFTS(T) = TFTS(T) + TFT

DESCRIPTION:  BUILDS THE TRANSITION FIRING TEMPLATE AS ATTRIBUTE
OF TRANSITION T.
```

Module: VERIFY.TPN

Module Description: This module allows the user to verify that the TEPN structure built is the same as that which was actually intended by the user. The module allows the user to examine either a single node within the net or the entire net.
Impact upon other Modules: Calls functions from other modules.
Data Structure unique to Module: none
Function Description:

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: DEFINE.TPN MODULE: VERIFY

```
        FUNCTION:  VERIFY.PLA(PNAME)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  PLACE-NAME PNAME
                EFFECT:  NONE

DESCRIPTION:  RETURNS COMPREHENSIVE DESCRIPTION OF A SPECIFIED PLACE.



        FUNCTION:  VERIFY.TRA(TNAME)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  TRANSITION-NAME TNAME
                EFFECT:  NONE

DESCRIPTION:  RETURNS COMPREHENSIVE DESCRIPTION OF SPECIFIED TRANSITION.



        FUNCTION:  VERIFY.NET(NETNAME)
                POSSIBLE VALUES:  NON
                PARAMETERS:  NETNAME NETNAME
                EFFECT:  NONE

DESCRIPTION:  RETURNS TEPN STRUCTURAL DEFINITION.
```

Subsystem EXEC.NET

Module Implementation Specifications

List of Modules

Module	Description
INIT.TPN	Initializes TEPN for execution
MASTER.TPN	Master control for TEPN execution
PLACE.TPN	Place data structure
TRANSITION.TPN	Transition data structure
TOKEN.TPN	Token data structure
TEPN.TPN	General TEPN attributes (external)

<u>Module</u>: INIT.TPN

<u>Module Description</u>: Module initializes TEPN marking and other functions required prior to the execution of a TEPN.

<u>Impact upon other Modules</u>: Calls functions within other modules.

<u>Data Structure Unique to Module</u>: Net initialization format.

<u>Function Description</u>:

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET MODULE: INIT.TPN

```
        FUNCTION:  MARK.PLA.INI (PNAME,TOK)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  PLACE-NAME PNAME, TOKEN-ID TOK
                EFFECT:  P=PLACEID (PNAME)
                         CALL TOKAPP.PLA (P,TOK)
```

DESCRIPTION: INSERTS AN INITIAL TOKEN INTO A PLACE

```
        FUNCTION:  BLDTOK.INI (TOK,ATTRSET)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  TOKEN NAME TOK, TOKEN ATTRIBUTE
                             SET ATTRSET
                EFFECT:  NONE
```

DESCRIPTION: BUILDS AN INITIAL STATE TOKEN WITH USER SUPPLIED ATTRIBUTE SET.

```
        FUNCTION:  RUN.INI (PLIST)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  RUN PARAMETER LIST PLIST
                EFFECT:  INITIALIZE ALL RUN PARAMETERS
```

DESCRIPTION: THIS FUNCTION WILL ESTABLISH ALL THE MECHANISMS REQUIRED BY RUN PARAMETERS. SINCE THESE ARE IMPLEMENTATION CHARACTERISTICS AND DO NOT AFFECT THE TEPN, NO DESIGN IS PRESENTED IN THIS THESIS FOR PARAMETER OPERATION. AS A MINIMUM, THE MODEL HALT CRITERION IS NEEDED AS A PARAMETER.

Module: MASTER.TPN

Module Description: Master "control" module that keeps track of events that need to be monitored and which are important to the immediate operation of the net.

Impact Upon Other Modules: None

Data Structure Unique to Module: Future events list/chain.

Function Description:

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET MODULE: MASTER.TPN

```
        FUNCTION:  FEVSCH.MAS (P, TIME)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  PLACED-ID P;
                             INTEGER TIME.
                EFFECT:  NONE
```

DESCRIPTION: THIS FUNCTION QUEUES A FUTURE EVENT, THE TRANSFER OF A SPECIFIED TOKEN IN A SPECIFIED PLACE FROM THE PAQ TO THE FEQ AT AN ALREADY DETERMINED TIME.

```
        FUNCTION:  FEVREM.MAS (P, TIME)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  PLACE-ID P, INTEGER TIME
                EFFECT:  NONE
```

DESCRIPTION: DELETES AN EVENT FROM THE FUTURE EVENT QUEUE.

```
        FUNCTION:  NXTEV.MAS
                POSSIBLE VALUES:  NONE
                PARAMETERS:  NONE
                EFFECT:  IF HALT PARAMETERS MET THE HALT ELSE
                             P=P.EVT((QNEXT(FEVQ))
                           CALL PLAENA (P)
```

DESCRIPTION: IDENTIFIES THE NEXT EVENT IN THE FUTURE EVENTS QUEUE AND REMOVES THAT QUEUE ENTRY. PLAENA IS CALLED TO ENABLE THE TOKEN AT THE PAQ IN THE APPROPRIATE PLACE. THIS TRIGGERS ANY NEWLY ENABLED TRANSITIONS AND RESULTING CHANGES TO THE TEPN STATE.

Module: PLACE.TPN

Module Description: All functions concerning the internal characteristics of each place.

Impact Upon Other Modules: None

Data Structure Unique to Module: PLACE

Function Description:

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET MODULE: PLACE.TPN

```
        FUNCTIO:  PLACEID (PNAME)
                POSSIBLE VALUES:  PLACE-ID.
                PARAMETERS:  ANUM-STRING PNAME
                EFFECT:  NONE
                INITIAL VALUE:  NONE
                                 RETURNS THE INTERNAL PLACE-ID WHEN
                                GIVEN THE USER DEFINED PLACE NAME.

DESCRIPTION:



        FUNCTION:  BOUND.PLA(P)
                POSSIBLE VALUES:  NONEPOSITIVE INTEGERS
                PARAMETERS:  PLACE-ID P
                EFFECT:  NONE
                INITIAL VALUE:  HIGHVALUES
DESCRIPTION:  SUM OF THE PED AND PAD BOUNDS.



        FUNCTION:  PLAACT.PLA   (P,TOK)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  PLACE-ID P; TOKEN-ID TOK
                EFFECT:  CALL PADADD.PLA (P,TOK)


DESCRIPTION:  "ACTIVATES" PLACE UPON ARRIVAL OF A TOKEN FROM
                AN INPUT TRANSITION.
```

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET MODULE: PLACE.TPN

```
        FUNCTION:  QDISC.PEQ (P)
                POSSIBLE VALUES:  QUEUEING DISCIPLINES
                PARAMETERS:  PLACE-ID P
                EFFECT:  NONE
                INITIAL VALUE:  INFINITE PROCESSOR

DESCRIPTION:  THE PLACE ENABLED QUEUE IS DEFINED BY ONLY
TWO FUNCTIONS SINCE  THERE IS NO INTERNAL DELAY FUNCTION
APPLIED TO TOKENS ENTERING THE PEQ.
THE INITIAL QUEUEING DISCIPLINE IS "INFINITE PROCESSOR"
WHICH ALLOWS ALL TOKENS THAT HAVE COMPLETED THE INTERNAL
DELAY AT THE PAQ TO BE IMMEDIATELY AVAILABLE TO ENABLE
AND FIRE TRANSITIONS.

        FUNCTION:  BOUND.PEQ(P)
                POSSIBLE VALUES:  POSITIVE INTEGERS
                PARAMETERS:  PLACE-ID P
                EFFECT:  NONE
                INITIAL VALUE:  HIGHVALUE

DESCRIPTION:  QUEUE BOUND ON THE PEQ.



        FUNCTION  SIZE.PEQ (P)
                POSSIBLE VALUES:  POSITIVE INTEGERS
                PARAMETERS:  PLACE-ID P
                EFFECT:  NONE
                INITIAL VALUE:  0

DESCRIPTION:  SIZE OF THE PEQ, THE NUMBER OF TOKENS THERE.



        FUNCTION:  SIZE.PAQ (P)
                POSSIBLE VALUES:  POSITIVE INTEGERS
                PARAMETERS:  PLACE-ID P
                EFFECT:  NONE
                INITIAL VALUE:  0

DESCRIPTION:
```

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET MODULE: PLACE.TPN

```
        FUNCTION:  INTMARK.PLA(P)
                POSSIBLE VALUES:  SET OF TOKENS
                PARAMETERS:  PLACE-ID P
                EFFECT:  NONE
                INITIAL VALUE:  NULL SET

DESCRIPTION:  RETURNS THE INTERNAL MARKING, THE SET OF ALL
TOKENS IN A PLACE.


        FUNCTION:  PAQADD.PLA (P,TOK)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  PLACE-ID P; TOKEN-ID TOK
                EFFECT:  INTMARK (P) = INTMARK (P) + TOK
                         IF TOK = QNEXT.PLA(PAQ,P) THEN
                              TIME = TOTM.PLA
                            CALL FEVSCH.MAS (P,TIME)
                INITIAL VALUE:  N/A

DESCRIPTION:  ADDS INCOMING TOKEN, TOK, TO THE PAQ OF PLACE P.
              INTERNALLY, THE TOKEN WILL BE INSERTED ACCORDING
              TO THE PAQ'S QUEUEING DISCIPLINE.


        FUNCTION:  QNEXT.PLA (Q,P)
                POSSIBLE VALUES:  TOKEN
                PARAMETERS:  QUEUE Q; PLACE-ID P
                EFFECT:  NONE
                INITIAL VALUE:  NULL

DESCRIPTION:  RETURNS THE "NEXT" TOKEN, OR THE TOKEN "ON TOP"
SCHEDULED TO LEAVE THE QUEUE.  IF THERE IS MORE THAN ONE AVAILABLE
TOKEN THE FUNCTION WILL RETURN ONE CHOSEN AT RANDOM.
```

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET MODULE: PLACE.TPN

```
        FUNCTION:  TDTM.PLA (P,TOK)
                POSSIBLE VALUES:  INTEGER
                PARAMETERS:  PLACE-ID P; TOKEN-ID TOK
                EFFECT:  NONE
                INITIAL VALUE:  0

DESCRIPTION:  EXECUTES THE PLACE TDTM ON THE INCOMING TOKEN.
              THIS FUNCTION IS DEFINED BY THE USER AT NET
              DEFINTION TIME AND MAY USE THE ATRIBUTES OF THE
              TOKEN PAS IN ITS COMPUTATION.




        FUNCTION:  PLASEND.PLA (TOKTYSET,T)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  SET OF TOKEN-TYPES TOKTYSET;
                             TRANSITION T
                EFFECT:  ENATOK.PLA(TOKTYPE)
                         CALL PEQDEL.PLA (P,TOK)
                         CALL ADDINTOK.TRA (T,TOK)
                INITIAL VALUE:  N/A

DESCRIPTION:  TRIGGERS PLACE TO "SEND" A TOKEN OF ONE OF THE
              TYPES IN TOKTYPE TO TRANSITION T AS PART OF A
              TRANSITION FIRING.




        FUNCTION:  PAQDISC.PLA(P)]
                POSSIBLE VALUES:  QUEUE DISCIPLINE
                PARAMETERS:  PLACE-ID P
                EFFECT:  NONE
                INITIAL VALUE:  USER INPUT

DESCRIPTION:  USER SPECIFIES EACH PAQ QUEUE DISCIPLINE.
```

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET MODULE: PLACE.TPN

```
        FUNCTION:  PAQBND.PLA(P)
                POSSIBLE VALUES:  INTEGER
                PARAMETERS:  PLACE-ID P
                EFFECT:  NONE
                INITIAL VALUE:  HIGHVALUE

DESCRIPTION:  QUEUE BOUND (ONE FOR EACH PAQ).



        FUNCTION:  TOTMDEF.PLA(P)
                POSSIBLE VALUES:  FUNTION DEFINITIONS
                PARAMETERS:  PLACE-ID P
                EFFECT:  NONE
                INITIAL VALUE:  ZERO FUNCTION

DESCRIPTION:  THE VALUE RETURNED IS A USER DEFINED FUNCTION
WHICH USES THE FUNCTIONAL ATTRIBUTE SET AS ISTS PARAMETERS
AND RETURNS THE TOKEN DELAY TIME.



        FUNCTION:  ENATOK.PLA(TOKTYPE)
                POSSIBLE VALUES:  A TOKEN-ID/NULL
                PARAMETERS:  TOKEN TYPE TOKTYPE
                EFFECT:  NONE
                INITIAL VALUE:  NULL

DESCRIPTION:  SEARCHES THE PEQ. IF IT FINDS AN ENABLED
AND READY TOKEN MATHCING THE TYPE PASSED, IT PASSES
THAT TOKEN-ID.
```

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET MODULE: PLACE.TPN

```
        FUNCTION:  REQDEL.PLA(TOK,P)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  TOKEN-ID TOK, PLACE-ID P
                EFFECT:  INTMARK(P) = INTMARK(P) - TOK
                INITIAL VALUE:  N/A

  DESCRIPTION:  DELETES A TOKEN FROM THE REQ.


        FUNCTION:  EXTMARK.PLA(P)
                POSSIBLE VALUES:  SET OF ORDERED PAIRS (TOKTYPE,N)
                     TOKETYPE IS A TOKEN TYPE, N IS A NON-NEGATIVE
                   INTEGER
                PARAMETERS:  PLACE-ID  P
                EFFECT:  NONE
                INITIAL VALUE:  NULL

  DESCRIPTION:  RETURNS A SET OF ORDERD PAIRS (TOKEN TYPE,
  NUMBER OF TOKENS OF THIS  TYPE AT PLACE P).


        FUNCTION:  SETTIM.PLA(P,I)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  PLACE-ID P,INTEGER I
                EFFECT:  TIME(P) = I
                INITIAL VALUE:  N/A

  DESCRIPTION:  SET THE PLACE INTERNAL CLOCK TO A CERTAIN TIME.
```

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET　　　　MODULE: PLACE.TPN

```
        FUNCTION:  TICKTIM.PLA(P)
                POSSIBLE VALUES:  NONE
                EFFECT:  TIME(P) = TIME(P) -1
                PARAMETERS:  PLACE-ID P
                INITIAL VALUE:  N/A

DESCRIPTION:  DEFINES A DECREASING PLACE CLOCK FUNCTION.


        FUNCTION:  ERTOK.PLA(TOK,P)
                POSSIBLE VALUES:  TRUE OR FALSE
                PARAMETERS:  TOKEN-ID TOK,PLACE-ID P
                EFFECT:  NONE
                INITIAL VALUE:  FALSE

DESCRIPTION:  RETURNS TRUE IF TOK IS ENABLED AND READY IN
THE PEQ.


        FUNCTION:  PLAFIRE.PLA(P)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  PLACE-ID P
                EFFECT:  FOR EACH MEMBER T OF POUTSET.TRN(P)
                                CALL TRAFIRE.TRA(T)

DESCRIPTION:  FOR ALL ARCS OUT FROM A PLACE TO A TRANSITIONS
CALL A FUNCTION WHICH FIRES THAT TRANSITION IF IT IS NOW
ENABLED.


        FUNCTION:  PEQADD.PLA(P,TOK)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  PLACE-ID P, TOKEN-ID TOK
                EFFECT:  IF ERTOK.PLA(TOK,P)
                            CALL PLAFIRE.PLA(P)
DESCRIPTION:  ADDS A TOKEN TO THE PEQ AND UPDATES THE QUEUE.
IF THERE IS ANY CHANGE TO THE SET OF ENABLED ACTIVE TOKENS
FUNCTIONS ARE CALLED WHICH CHECK ALL OUTGOING ARCS FROM
THIS PLACE AND FIRE ANY NEWLY ENABLED TRANSITIONS.
```

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET MODULE: PLACE.TPN

```
        FUNCTION:  QSIZE.PLA(Q,P)
                POSSIBLE VALUES:  POSITIVE INTEGER
                PARAMETERS:  LOCAL QUEUE ID Q, PLACE-ID P
                EFFECT:  NONE
                INITIAL VALUE:  NONE

DESCRIPTION:  RETURNS THE SIZE OF A QUEUE.



        FUNCTION:  PAQDEL.PLA(P)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  PLACE-ID P
                EFFECT:  TOK = QNEXT.PLA(PAQ,P)
                         CALL PEQADD.PLA (P,TOK)
                         IF QSIZE.PLA(PAQ,P) > 0
                            TIME = TDTM.PLA(P,TOK)
                            CALL FEVSCH.MAS(P,TIME)
                INITIAL VALUE:  N/A

DESCRIPTION:  TRANSFERS QNEXT (PAQ,P) TO THE PEQ BY CALLING
PEQADD.  THEN THE PAQ IS REVIEWED AND A FUNCTION CALLED
TO INSERT AN ENTRY IN THE FEVQ IF THE PEQ IS NOT EMPTY.



        FUNCTION:  PLAENA.PLA(P)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  PLACE-ID P
                EFFECT:  CALL PAQDEL.PLA(P)
                INITIAL VALUE:  N/A

DESCRIPTION:  TRANSFERS THE NEXT TOKEN ON THE PAQ TO THE PEQ,
THEN CALLS FUNCTIONS WHICH UPDATE THE PAQ AND PEQ AND MAKE
RESULTANT STATE CHANGES TO THE TEPN.
```

AD-A105 381 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH F/G 9/2
TIME-EXTENDED PETRI NETS.(U)
AUG 79 F B BERLIN
UNCLASSIFIED AFIT-CI-79-205T NL

2 OF 2
AD
A105381

END
DATE
FILMED
10-81
DTIC

<u>Module</u>: TRANSITION.TPN

<u>Module Description</u>: Contains all functions necessary to "execute" the TRANSITION data structure.

<u>Impact Upon Other Modules</u>: None

<u>Data Structure Unique to Module</u>: TRANSITION

<u>Function Description</u>:

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET   MODULE: TRANSITION.TPN

```
FUNCTION:  TRAENA.TRA(T,TFT)
        POSSIBLE VALUES:  BOOLEAN (TRUE,FALSE)
        PARAMETERS:  TRANSITION-ID T, FIRING TEMPLATE TFT
        EFFECT:  NONE
        INITIAL VALUE:  FALSE
```

DESCRIPTION: CHECKS ALL OF TRANSITION T INPUT PLACES AGAINST TRANSITION FIRING TEMPLATES.
IF EACH PLACE HAS THE TOKEN REQUIRED TO ENABLE THE TRANSITION, THE FUNCTION RETURNS THE 'TRUE' AND ALSO RETURNS THE ENABLING FIRING TEMPLATE, TFT, FOR USE IN THE FIRING SEQUENCE (NOTE: THIS FIRING MUST BE AN INDIVISIBLE OPERATION.

```
FUNCTION:  TRAFIR.TRA(T)
        POSSIBLE VALUES:  NONE
        PARAMETERS:  TRANSITION-ID T
        EFFECT:  IF TRAENA(T,TFT) THEN
                      CALL PLASEND.PLA(TFT,T,INTOKSET)
                     CALL TTEXEC(T,INTOKSET)
```

DESCRIPTION: EACH TIME A TOKEN ENTERS THE "ENABLED-READY" STATE, THIS FUNCTION IS CALLED BY THE PLACE. IF A CALL TO TRAENA RETURNS 'TRUE', THE TRANSITION IS ENABLED AND TOKENS ARE COLLECTED FROM THE PLACES AND PASSED AS A SET TO TTEXEC.

```
FUNCTION:  ADDINTOK.TRA(T,TOK)
        POSSIBLE VALUES:  NOE
        PARAMETERS:  TRANSITION ID T, TOKEN-ID TOK
        EFFECT:  INTOKSET(T) = INTOKSET(T) + TOK
        INITIAL VALUE:  N/A
```

DESCRIPTION: ADDS A TOKEN TO THE END OF THE SET OF TOKENS COLLECTED FROM THE PLACES.

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET MODULE: TRANSITION.TPN

```
        FUNCTION:  TTEXEC.TRA(T,INTOKSET)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  TRANSITION-ID T,TOKEN SET INTOKSET
                EFFECT:  FOR EACH TT IN TTS(T)
                           TOK=NEWTOK
                           PAS(TOK)=TT(INTOKSET)
                           TOKTYPE(TOK) = TT(INTOKSET)
                           P=TTPLA.TRA(TT)
                           CALL TRASEND(TOK,P)
                INITIAL VALUE:  N/A

DESCRIPTION:  "EXECUTES" THE TRANSITION TOKEN TEMPLATE S,
CREATING THE NEW TOKEN AND CALLS A FUNCTION TO SEND THE
TOKEN TO THE APPROPRIATE PLACE.

        FUNCTION:  TTS.TRA(T)
                POSSIBLE VALUES:  SET OF TOKEN TEMPLATES
                PARAMETERS:  TRANSITION ID T
                EFFECT:  NONE
                INITIAL VALUE:  NULL SET

DESCRIPTION:  RETURNS SET OF TOKEN TEMPLATES.



        FUNCTION:  TTPLA.TRA(TT)
                POSSIBLE VALUES:  PLACE-ID
                PARAMETERS:  TOKEN-TEMPLATE ID TT
                EFFECT:  NONE
                INITIAL VALUE:  NONE

DESCRIPTION:  RETURNS THE PLACE-ID TO WHICH A TOKEN MUST BE SENT.
```

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET MODULE: TRANSITION.TPN

```
        FUNCTION:  TRASEND.TRA(TOK,P)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  TOKEN-ID TOK,PLACEID P
                EFFECT:  CALL PLAACT.PLA(P,TOK)
                INITIAL VALUE:  N/A

DESCRIPTION:  THE "COMMUNICATION INTERFACE" BETWEEN TRANSITION
T AND PLACE P, "SENDS" NEWLY CREATED TOKENS TO  PLACE P.



        FUNCTION:  TFTS.TRA(T)
                POSSIBLE VALUES:  SET FO TRANSITION FIRING TEMPLATES
                PARAMETERS:  TRANSITION T
                EFFECT:  NONE
                INITIAL VALUE:  NULL SET

DESCRIPTION:  SET OF TRANSITION FIRING TEMPLATES.
```

Module: TEPN.TPN

Module Description: Localizes functions concerned with the overall TEPN structure and which require manipulation of inter-nodal arcs.

Impact Upon Other Modules: none

Data Structures Unique to Module: TEPN

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET MODULE: TEPN.TPN

```
        FUNCTION:  INTMARK.TPN(TEPN)
                POSSIBLE VALUES:  TEPN INTERNAL MARKING
                PARAMETERS:  TEPN-ID TEPN
                EFFECT:  NONE
                INITIAL VALUE:  NONE
```

DESCRIPTION:

```
        FUNCTION:  EXTMARK.TPN(TEPN)
                POSSIBLE VALUES:  TEPN EXTERNAL MARKING
                PARAMETERS:  TEPN-ID TEPN
                EFFECT:  NONE
                INITIAL VALUE:  N/A
```

DESCRIPTION: RETURNS THE CURRENT EXTERNAL MARKING OF THE TEPN. THIS FUNCTION MAY ALSO BE USED AT THE END OF THE RUN TO OUTPUT THE FINAL MARKING.

```
        FUNCTION:  PERFSTAT.TPN(TEPN)
                POSSIBLE VALUES:  NONE
                PARAMETERS:  TEPN-ID TEPN
                EFFECT:  NOND
                INITIAL VALUE:  N/A
```

DESCRIPTION: RETURNS THE CURRENT OR FINAL PERFORMANCE STATISTICS FROM A TEPN RUN.

```
        FUNCTION:  TINFN.TPN(T)
                POSSIBLE VALUES:  SET OF PLACES
                PARAMETERS:  TRANSITION-ID T
                EFFECT:  NONE
                INITIAL VALUE:  NULL SET
```

DESCRIPTION: CORRESPONDS TO THE TRANSITION INPUT FUNCTION DEFINED BY THE BASIC TEPN DEFINITION AND RETURNS THE SET OF INPUT PLACES TO TRANSITION T.

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET MODULE: TEPN.TPN

```
        FUNCTION:  TOUTPN.TPN(T)
                POSSIBLE VALUES:  FUNCTION FROM TEPN TRANSITIONS
                    TO SETS OF TEPN PLACES
                PARAMETERS:  TRANSITION-ID T
                EFFECT:  NULL SET
                INITIAL VALUE:

DESCRIPTION:  CORRESPONDS TO THE TRANSITION FUNCTION DEFINED
BY THE BASIC TEPN DEFINITION  AND RETURNS THE SET OF OUTPUTS
PLACES FROM TRANSITION T.


        FUNCTION:  POUTSET.TPN(P)
                POSSIBLE VALUES:  SET OF TRANSITIONS
                PARAMETERS:  PLACE-ID P
                EFFECT:  NONE
                INITIAL VALUE:  NULL

DESCRIPTION:  GIVEN A PLACE-ID, RETURNS THE SET OF ALL TRANSITIONS
TO WHICH OUTWARD ARCS FROM THAT PLACE CONNECT.


        FUNCTION:  PLASET.TPN(TEPN)
                POSSIBLE VALUES:  SET OF PLACES
                PARAMETERS:  TEPN-ID TEPN
                EFFECT:  NONE
                INITIAL VALUE:  NULL SET

DESCRIPTION:


        FUNCTION:  TRASET.TPN(TEPN)
                POSSIBLE VALUES:  SET OF TRANSITIONS
                PARAMETERS:  TEPN-ID TEPN
                EFFECT:  NONE
                INITIAL VALUE:  NULL SET

DESCRIPTION:
```

Module: TOKEN.TPN

Module Description: Contains all functions necessary to build and delete tokens.

Impact Upon Other Modules: None

Data Structures Unique to Module: TOKEN

Function Description:

TIME-EXTENDED PETRI NET MODEL IMPLEMENTATION SPECIFICATIONS

SUBSYSTEM: EXEC.NET MODULE: TOKEN.TPN

```
        FUNCTION:  NEWTOK.TOK
                POSSIBLE VALUES:  TOKEN-ID
                PARAMETERS:  NONE
                EFFECT:  NONE
                INITIAL VALUE:  N/A

DESCRIPTION:  CALLED TO CREATE A NEW TOKEN STRUCTURE AND
RETURN THE INTERNAL-ID.


        FUNCTION:  FAS.TOK(TOK)
                POSSIBLE VALUES:  TOKEN FUNCTIONAL ATTRIBUTE SETS
                PARAMETERS:  TOKEN-ID TOK
                EFFECT:  NONE
                INITIAL VALUE:  NULL SET

DESCRIPTION:  RETURNS THE TOKEN FUNCTIONAL ATTRIBUTE SET (FAS)


        FUNCTION:  TYPE.TOK(TOK)
                POSSIBLE VALUES:  TOKEN TYPES
                PARAMETERS:  TOKEN-ID TOK
                EFFECT:  NONE
                INITIAL VALUE:  NONE

DESCRIPTION:  GIVEN A TOKEN-ID, RETURN THE TOKENTYPE
   NOTE THAT NO ACTIONS OR EFFECTS RESULT FROM
THE TOK MODULE, IT MERELY CREATES AND HOLDS AND REPORTS ATTRIBUTES
OF TOKENS.


        FUNCTION:
                POSSIBLE VALUES:
```

Chapter VI

A TEPN Model of a Disk Input/Output Subsystem

This chapter describes the application of the TEPN in modeling the performance of a complex disk input/output subsystem. Though this particular problem may, at first blush, appear as a trivial problem not significant in demonstrating the TEPN's potential for more general application, careful analysis shows that this case study, which was motivated by a real rather than hypothetical problem, requires a modeling methodology which meets the goals indicated in the Introduction to this Thesis. In particular, the disk input/output system described below underscores the need for a modeling methodology which can faithfully represent deterministic behavior, process blocking, and the holding of multiple resources by a process.

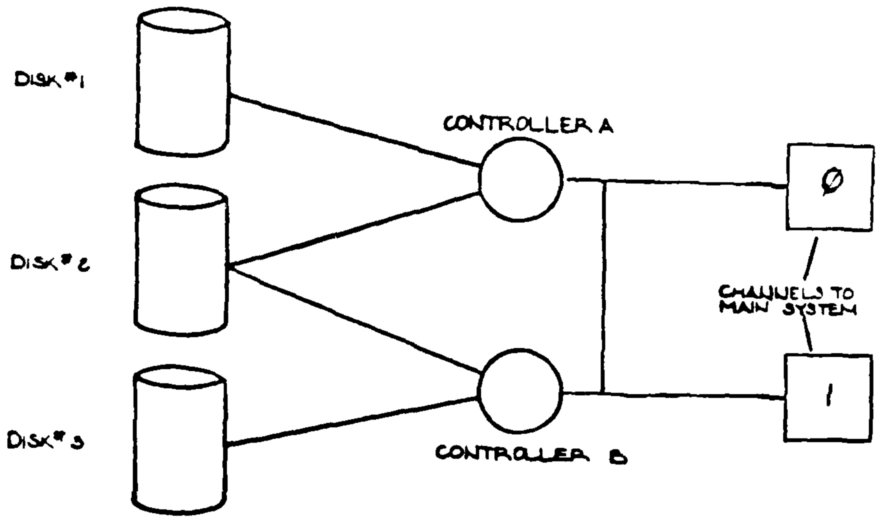
In describing the TEPN application, the chapter demonstrates the TEPN to be both useful and significant: useful in that it has sufficient modeling power to model many complex systems, and significant in that it can model systems which cannot be faithfully modeled by the queueing network model.

The chapter is organized into three major sections. The first describes the basic disk I/O configuration used throughout the chapter and the modeling problems that emanate from this configuration. The second section shows why the queueing network model is structurally inadequate for modeling the exact structure of the disk subsystem when either disk sharing or parallel path allocations are employed. The final section presents TEPN models of the serial and parallel implementations of the disk I/O configuration under study. These sections demonstrate that TEPN models can represent both the serial and the parallel cases.

Description of the Disk Subsystem Problem to Be Modeled

Subsystem Configuration

Figure 6-1 illustrates the configuration of a small disk subsystem representative of the type of disk input/output subsystem



DISK INPUT/OUTPUT SUBSYSTEM

FIGURE 6-1

configured to many data processing systems. Though the configuration illustrated is very small, it is nonetheless representative of many larger disk subsystems in that the same conceptual difficulties are encountered in modeling the smaller configuration as are encountered in modeling the larger configuration.[8]

In describing this subsystem for modeling purposes, it is necessary to describe both the physical configuration characteristics and the operational characteristics. The physical configuration characteristics define the structure, or, by analogy, the syntax of the system to be modeled. From Figure 6-1, we can note the following physical attributes:

(1) the channels are crossbarred across both controllers, with the result that either controller may be accessed by either channel;

(2) each controller is attached to a finite and fixed subset of the set of disk units and is part of a unique path from either of the channels to any disk units not shared with another controller;

[8] A "conceptual" difficulty relates only to modeling power/capability. We believe the TEPN is theoretically equipped to model any real disk I/O subsystem configuration. Whether a very large system can be "physically" modeled depends more upon the limits of the implementation of the model, and not the model itself.

(3) some disk drives may be shared by two or more controllers (in this case, the second drive is shared) with the result that there are multiple paths to each shared unit.

Controller/Disk Path Allocation Algorithm

In addition to the above characteristics, all of which deal with the physical configuration of the disk subsystem, there is another important, though diagrammatically invisible feature: the controller/disk unit path allocation algorithm, or the algorithm used to determine how to allocate and deallocate the controllers and disk units when handling input/output requests.

There are two major path allocation algorithms in use. The first, the serial algorithm, allocates an entire controller/disk unit path to a transaction from the time the request is approved for servicing until the transfer is complete. In this case, if a disk unit has to perform a seek operation before the data transfer operation can begin, the controller will initiate the seek and wait for it to finish, not accepting any other work during the idle period.

The parallel, or SEEK:READ/WRITE overlap algorithm, on the other hand, only allocates devices when they are specifically required for a transfer or seek function. Once the controller has initiated the seek operation, for example, the controller will be available to service other transfer requests while it is waiting for the disk to complete its seek operation. When the disk seek is completed, the controller must be reallocated to complete the path required for the actual transfer operation. Finally, when the operation is complete, both units will be returned to the system for allocation to other incoming and pending input/output requests.

Difficulties Encountered When Modeling the Disk Subsystem with the Queueing Network Model

The limitations of the queueing network model were discussed in Chapter II. This section presents some of these limitations as they particularly apply to the modeling of the disk subsystem problem discussed in this chapter.

Browne, et al [Browne, et al, 1973] describe the representational problems inherent to modeling a complex disk subsystem (i.e., other than the trivial cases of single-controller, single-disk systems) in which disk sharing and/or the parallel path allocation algorithm are employed.[9] In particular, they showed that the queueing network model cannot model the exact system interactions and that when these interactions are crucial to the model it is currently necessary to model and simulate the subsystem using a discrete simulation language model. Three of the difficulties which they cited are described below.

[9] Browne, et al, studied performance characteristics of a large Control Data Corporation (CDC) system which included disk drives especially equipped with the overlap algorithm.

Process Blocking. There are two types of blocking which may happen with a system such as the one above. The first involves simple queueing on the controllers, called primary blocking. In this case, both of the controllers are busy, either initiating a seek or performing a data transfer operation. In either case, the request will be queued on the incoming path, and will have no effect upon the disk units. Secondary blocking may occur when a controller initiates a seek operation in a disk unit and goes on to service another request, and is not free to begin a transfer when the seek is completed, and no alternate paths are free. Intuitively, every time that the overlap feature is utilized there is an opportunity for secondary blocking to occur.

Holding of Multiple Resources. During its life cycle, each disk request transaction must hold each of three resources: a channel, a controller, and a disk drive. Although the queueing network model cannot directly model the holding of multiple resources if the resources are always deallocated in the exact reverse order of their original allocation, a queueing network model can be designed which will faithfully model the system behavior. Such is the case with the serial path allocation algorithm. If the parallel path allocation algorithm is used, however, the controller and channel may be deallocated and then reallocated in the middle of a transaction's life cycle. This process cannot be faithfully modeled with a queueing network model.

Determinism and Multiple Configurations. The need for a deterministic model versus a stochastic representation is difficult to see in some cases in which the workload and system characteristics are such that the stochastic model closely approximates the real system. However, when either fine-grain accuracy is important, the workload characteristics are not easily "summarized" into stochastic functions, or the configuration may change, a discrete, deterministic methodology is clearly more desirable and sometimes essential. When fine-grain accuracy is important, the deterministic model allows the analyst to model exact hardware characteristics, rather than average service times. This may be particularly useful, for example, for studying different disk storage strategies (such as only using a certain block of tracks on each disk drive). One particular advantage of a deterministic model is that it may more easily be trace-driven, thus reducing the need to preanalyze and characterize the input workload. Finally, when studying a variety of configurations, it is sometimes important to have a model which would exactly quantify the impact of secondary effects (such as the increase or decrease in secondary blocking) before the effects are understood enough to characterize them stochastically.

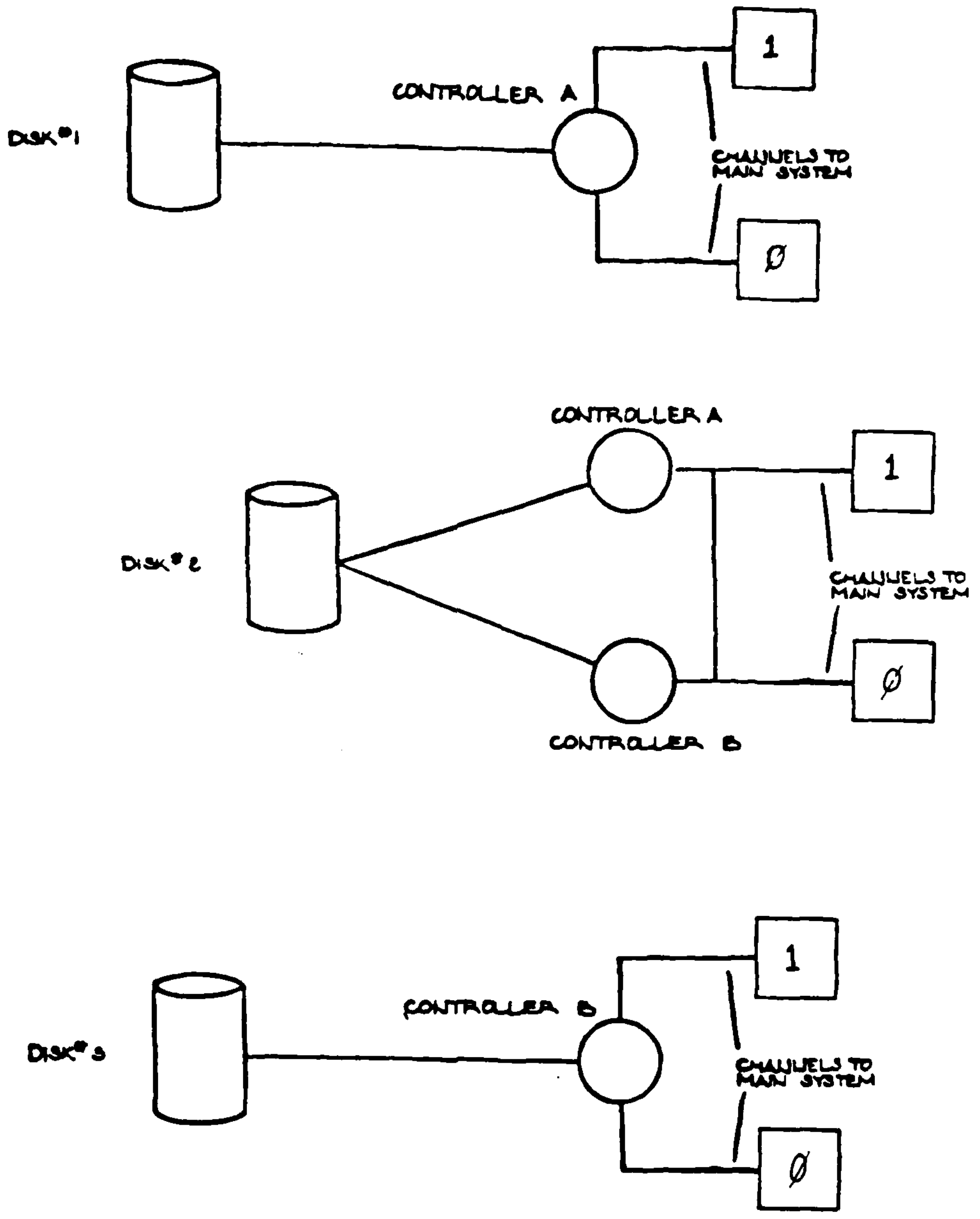
TEPN Model of the Disk Subsystem with Serial Path Allocation

The description of the TEPN model of the disk subsystem with serial path allocation is divided into three sections. The first section deals with the model structure and configuration. The second section builds upon that structure by defining the TEPN model parameters which are characteristic of this particular model. The final section concerned with the TEPN model of the disk subsystem is a brief discussion of the model's operational characteristics.

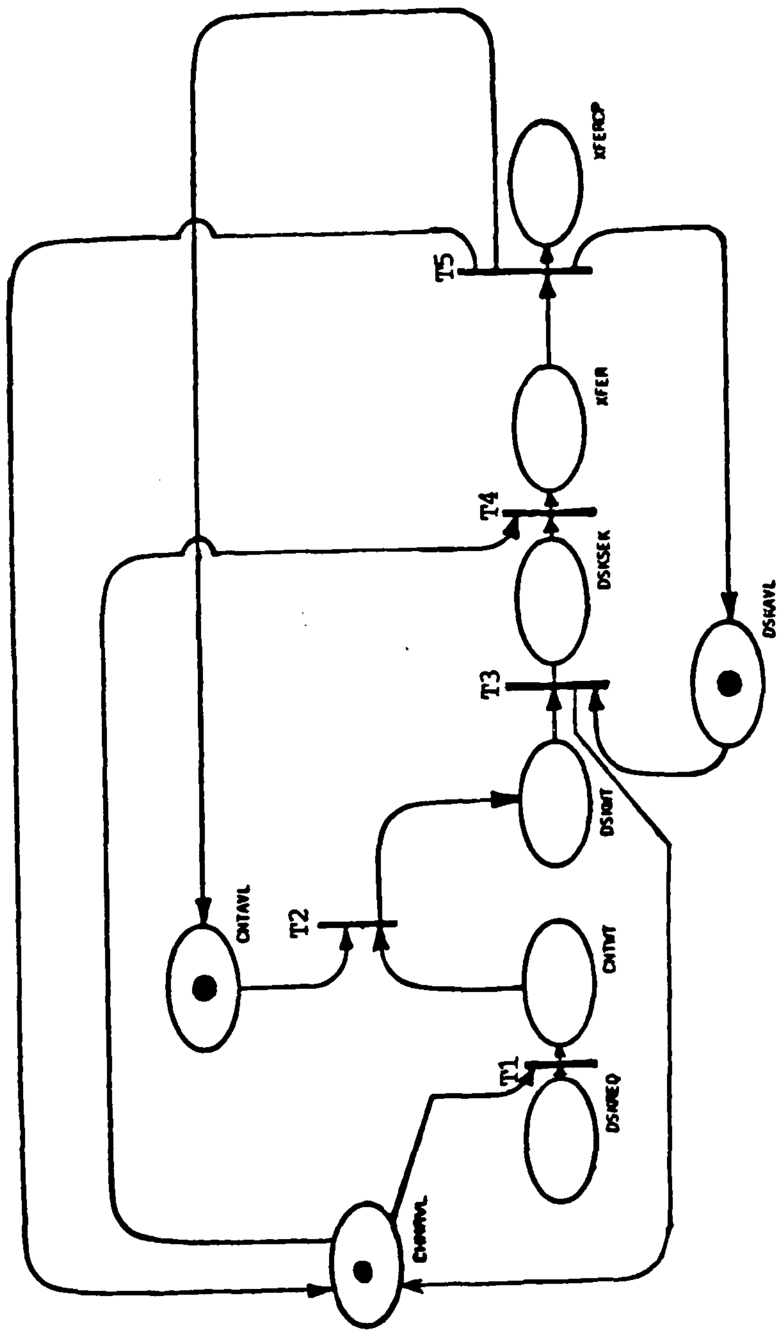
Once a model has been defined for the serial path allocation case, only a few modifications are required to convert the model to represent the same disk subsystem with parallel path allocation. These modifications and the final model are presented in the chapter's final section.

Disk Subsystem Structure and Organization

The TEPN model structure for the disk subsystem was designed in three stages. The first stage was to decompose the two-controller, three-disk subsystem into three separate disk subsystems, each of which contains the data paths possible for transferring data to and from one of the three disk drives. This system decomposition is diagrammatically illustrated in Figure 6-2. From Figure 6-2 we can see that the two controller three-drive disk subsystem can be modeled as a combination of two single-controller, single-disk subsystems combined with one dual-controller, single-drive subsystem. Figures 6-3 and 6-4 illustrate marked TEPN model structures which could effectively represent these two basic configurations. The reader should note that the structure shown in Figure 6-4 is essentially identical to that shown in Figure 6-3, with the exception that a controller may be

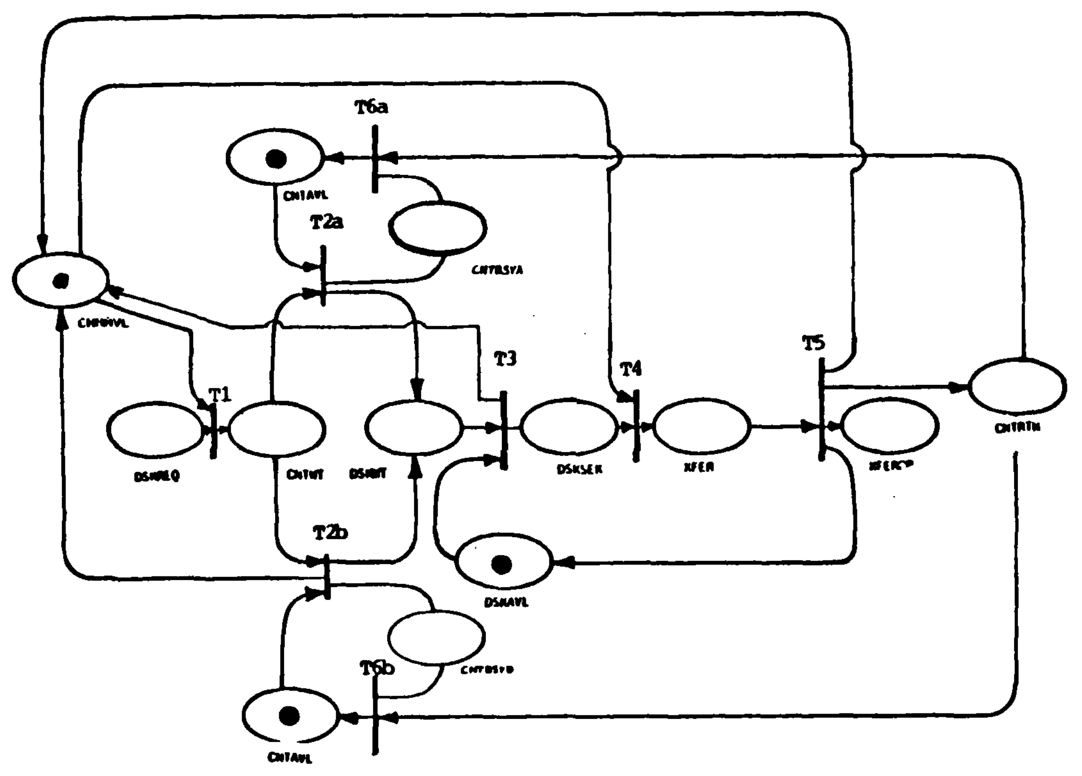


DISK SUBSYSTEM CONTROLLER/DISK PATH DECOMPOSITION
FIGURE 6-2



Marked TEPN Model of 1-Controller, 1-Disk Subsystem (Serial Path Allocation)

Figure 6-3



MARKED TEPN MODEL OF 2-CONTROLLER, 1-DISK SUBSYSTEM (SERIAL PATH ALLOCATION)

FIGURE 6-4

drawn from one of two nodes named CNTAVL instead of the single CNTAVL node shown in Figure 6-3. The addition of the CNTBSY places insures that the token is returned to one of two CNTAVL nodes after the data transfer has been completed. Table 6-1 presents the interpretation of each of the places labeled in Figure 6-3, Figure 6-4, and other figures throughout Chapter VI; Table 6-2 presents similar information for each of the transitions.

In order to model the entire two-controller, three-disk system, we can combine the models of the separate parts of the subsystem into a single large model. This model is shown in Figure 6-5. It is important to realize that this illustration only shows the structure of the model. From this structure we can determine only what "states" are possible ("reachable") or other analysis possible with a normal Petri net (since in the default case the TEPN is equivalent to a Petri net). It is not possible, however, to examine the model's time-resolved performance without introducing TEPN semantic (as opposed to structural or syntactic) attributes.

Although the model in Figure 6-5 represents the disk subsystem, one can quickly observe that a model built in this manner would rapidly become very large and very cumbersome for all but the simplest disk subsystems. Therefore, it is necessary to be able to reduce or "collapse" this large structure into a simpler but equivalent structure. An appropriate reduction method has been suggested by both Petri net researchers and the TEPN definition.

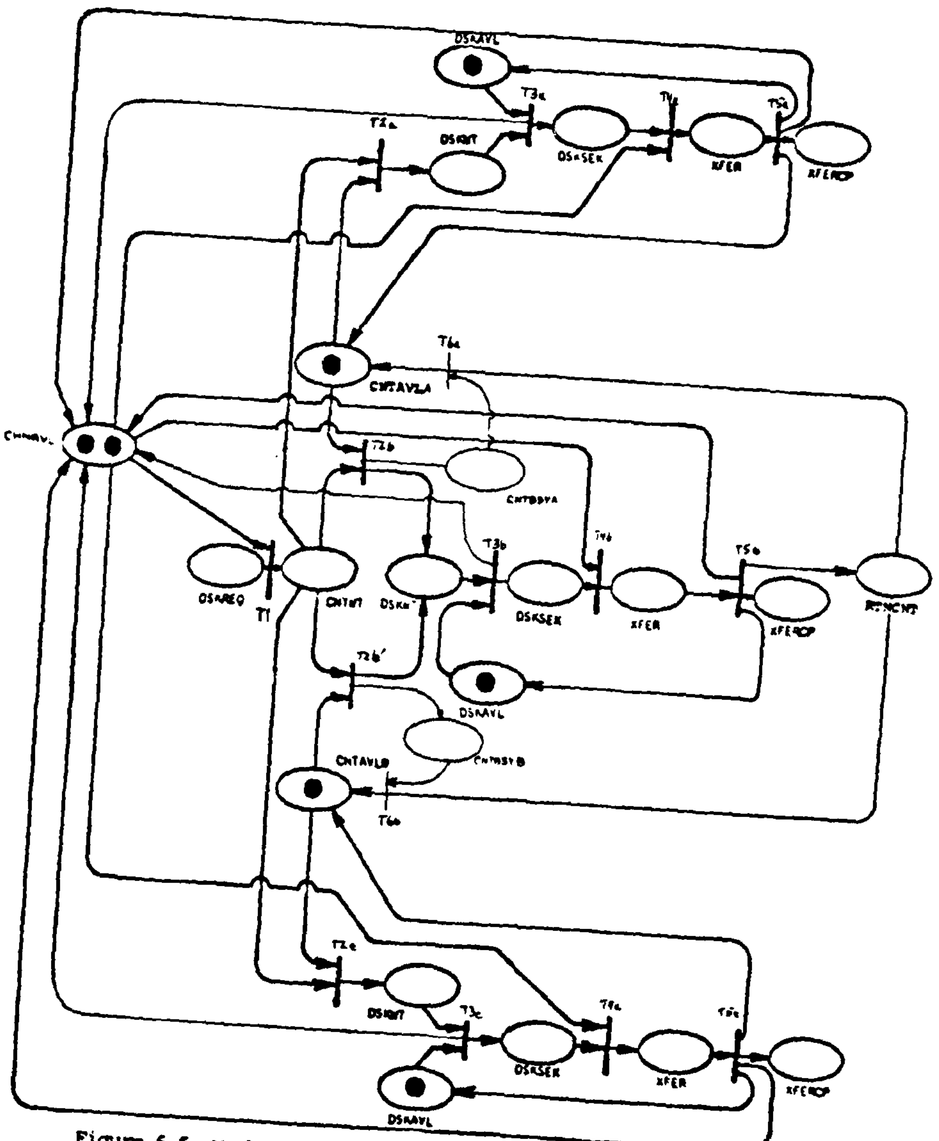


Figure 6-5 Marked TEPN Model of 2-Controller, 3-Disk I/O Subsystem

PLACE NAME	PLACE INTERPRETATION/DESCRIPTION
DSKREQ	A token at this place indicates that request for a data transfer is waiting in the system.
CNTWT	A token at this place indicates that the model or the subsystem being modeled is in a wait state waiting for a controller to become available.
CNTAVL	The tokens at this place indicate the availability of a controller in the system. In the cases of the models in Figures 6-3 and 6-4 these places are bounded to a maximum of one token. That token then becomes representative of the exact controller for which the place is designated.
DSKWT	When there is both a token at place CNTWT and CNTAVL, transition T1 will fire and transfer a token to place DSKWT. This place indicates that the model is in a wait state for an available disk drive. In the case of these models, of course, this disk drive is exactly specified to be either disk number one, number two or number three.
DSKAVL	Tokens residing in this place indicate that a disk drive is available for matching with the token at DSKWT or in terms of the actual system being modeled, a disk drive is available for matching with the appropriate disk request.
DSKSEK	Disk seek time is composed of both track seek time and rotational delay.

DESCRIPTIONS OF PLACES WITHIN
TEPN MODEL OF DISK I/O SUBSYSTEM

TABLE 6-1

PLACE NAME	PLACE INTERPRETATION/DESCRIPTION
XFER	The token at this place indicates that a data transfer is under way and therefore that the disk/controller path is in active state.
XFERCP	A token at this place indicates that the transfer is complete. Note that the same transition that sends a token to this place also sends a token to both CNTAVL and DSKAVL, thus "deallocating" both the controller and the disk that was in use during the transfer operation.
*XFERWT	For each request being processed by the disk subsystem, there is a token resident at this place. When the transfer is completed, the associated token is released with the result that the local wait time of the XFERWT token is exactly equal to the combined total wait times for all aspects of handling the request.
CHNAVL	A token at this place indicates channel availability.
CNTRTN	This place is only included in the model illustrated in Figure 6-4 and is required in order to allow a token to return to either one of the CNTAVL places. In terms of the actual system, this would allow the controller to be deallocated to the proper place. If controller A, for example, had been allocated as part of the original transfer path, then the token at CNTRTN would travel through to the transition which has as its output place the CNTAVL associated with controller A; and, similarly, if controller B had been part of the allocated path, at deallocation time the CNTAVL place associated with controller B would receive the token.
**CNTBSY	The two places with this name (CNTBSYA and CNTBSYB) match with place CNTRTN to insure that the token in CNTRTN fires the correct CNTAVL place.

*Not included in Figures 6-3, 6-4, and 6-5 but included in the final models of the disk subsystem.
**Only included in Figures 6-4 and 6-5.

DESCRIPTIONS OF PLACES WITHIN
TEPN MODEL OF DISK I/O SUBSYSTEM

TABLE 6-1 (Cont'd)

TRANSITION	INTERPRETATION OF FIRING
T1	[Any] Channel allocated to queued disk request; transaction processing cycle begins.
T2	[Specified] Controller allocated to queued disk request transaction; disk request queued on [specified] disk.
T3	[Specified] Disk allocated to queued disk request transaction; disk seek activated; channel deallocated pending completion of disk seek operation.
T4	[Any] Channel allocated for disk I/O operation; disk I/O operation initiated.
T5	End of disk request transaction life cycle; channel, controller, and disk unit deallocated.
*T6	[Specified] Controller returned to available ("idle") state.

*only included in Figures 6-4 and 6-5.

DESCRIPTIONS OF TRANSITIONS
TEPN MODEL OF DISK I/O SUBSYSTEM

TABLE 6-2

Within the Petri net, the reduction is based upon properties of token types or colors. Research concerning colored petri nets was mentioned earlier in this Thesis (see footnote 6, page 43). With the TEPN, this reduction capability exists in the form of transition firing templates and the companion token types and token templates. Using the firing template concept, the model of Figure 6-5 can be collapsed into a compact model which can represent a disk subsystem of virtually arbitrary size and configuration complexity. The parameters necessary to form this reduction are presented in the next section.

TEPN Disk Subsystem (Serial Path Allocation) Model Parameters

This section presents the TEPN model parameters which are required to model the performance of the disk subsystem presented in Figure 6-1 of this chapter. These parameters define not only the characteristics of the controller disk interactions but are also used to define the actual configuration of the disk subsystem. The place attributes are primarily concerned, of course, with time resolution, and therefore embody the attributes which impact such things as disk seek time and data transfer times. These attributes are the same regardless of the number of disk units or controllers within the subsystem. They are also insensitive to the configuration of these disks or controllers. The transition firing templates are used in conjunction with the token types to define the configuration of the disk subsystem. In particular, the transition firing templates are used to insure that only legal controller disk paths are allocated. Without the firing template, for example, any controller token could be matched with any disk token to enable a transition which would eventually result in a data transfer. While this might be acceptable in some systems, it is not acceptable in a trace-driven modeling environment where the specific disk units are paired with disk requests and the performance of individual disk drives and controllers is of concern to the analyst.

Transition Firing Template. The Transition Firing Template determines the configuration of the disk subsystem by controlling which controller tokens will match with which disk or disk specific tokens while the net is in operation. Within the model of Figure 6-5, the major reason for firing templates is to allow a method of matching disk requests with the appropriate disk drive or unit and matching disk units with controllers that can service them. For example, if a disk request arrives for a transfer of data that is residing on, in the real system, disk unit number two, we know that that disk unit can be accessed through either controller A or controller B. However, if another disk request arrives that must access disk unit number one, this disk unit must only be paired with controller A. In Figure 6-5, this problem is handled by having a separate model for each possible controller/disk combination. However, as mentioned before, such an approach would require a very cumbersome network in even a small subsystem. A firing template that requires that the input token from CNTAVL match a certain type token from CNTWT, however, can be used to guarantee that only the appropriate controller is matched with the disk request. It is thus possible to "collapse" all of the models for various controller/disk paths into a single model. Therefore, the first firing template to be described describes the match between controllers requests and disk units. Table 6-3 describes the firing templates required to model the actual configuration shown in Figure 6-1, with serial path allocation; Figure 6-6 shows the new model structure that results when these firing templates have been implemented. Finally, Table 6-4 presents the formal model structure definition (in terms of Definition 4.1) of the structure in Figure 6-6.

Place Attributes. The place attributes are divided into three main groups: the attributes of the clock, the attributes of the Place Active Queue (PAQ), the attributes of the Place Enabled Queue (PEQ), and the set of Place Performance Functions (PPF). Table 6-5 lists the values of each of these attribute groups as required for modeling the disk subsystem with serial path allocation. Since each of the attributes were explained in Chapter IV, the table is presented with little additional explanation, beyond the keys located at the bottom of the chart which define parameters used. The functions used to define the TDTM place attributes are explained in Table 6-6.

TRANSITION	INPUT PLACES	TOKEN TYPE DOMAIN	FIRING TEMPLATE GOAL	ENABLING FIRING TEMPLATES

T1	DSKREQ
	CHNAVL
		{1,2,3}
		{Ø,1}
			Match any channel with disk request	({1,2,3}, {Ø,1})

T2	CNTWT CNTAVL	{[{Ø,1}, {1,2,3}] {A,B}	Match disk request with proper controller	([*, {1,2}],A) ([*, {2,3}],B)

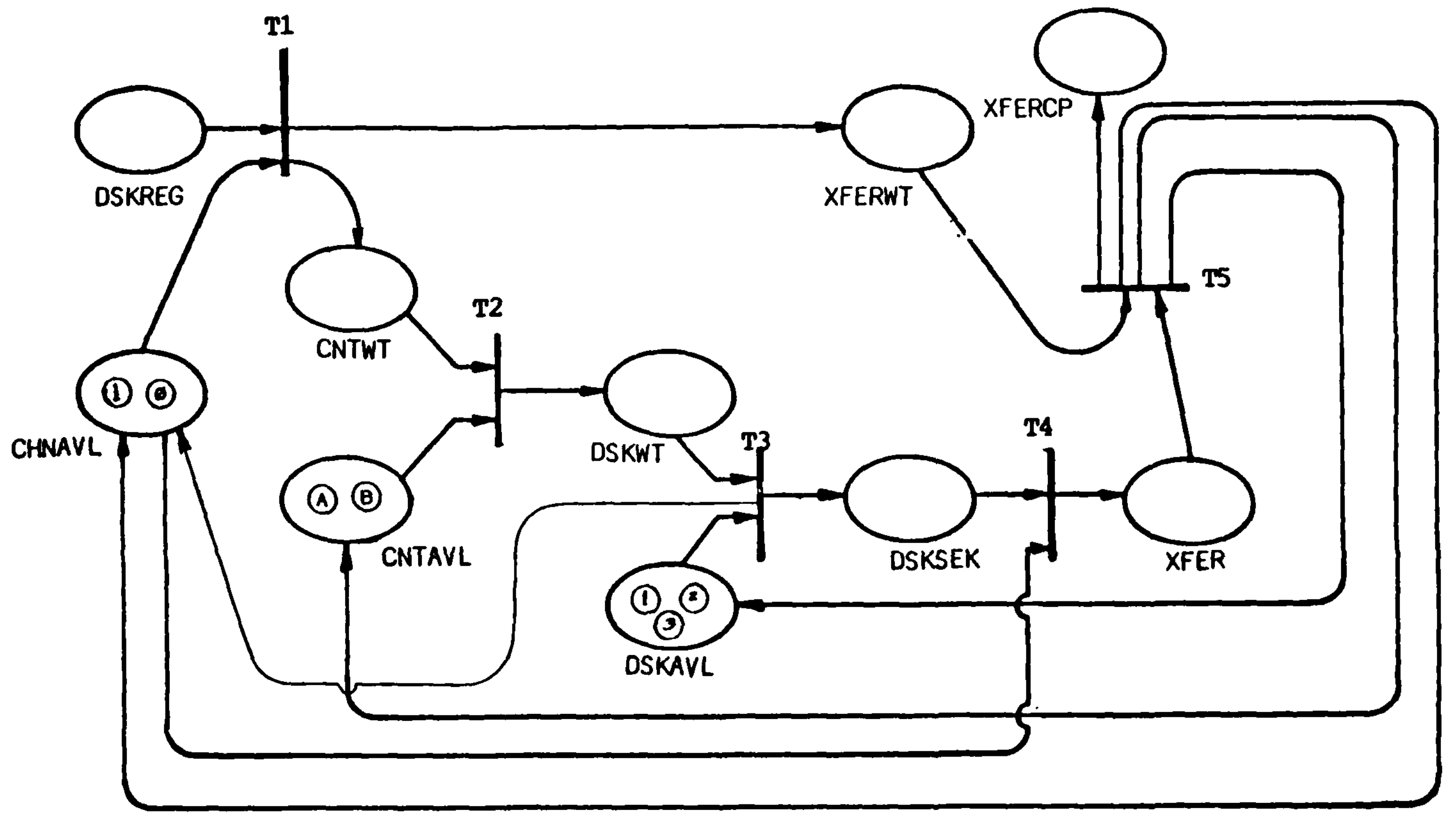
T3	DSKWT
	
	DSKAVL
		{[{Ø,1},{1,2},A],
		[{Ø,1}, {1,3},B]}
		{1,2,3}
			Match disk request with disk unit	([*,1,*],1)
				([*,1,*],2)
				([*,3,*],3)

T4	DSKSEK
	
	CHNAVL
		{[{1,2},A] , [{2,3},B]}
		
		{Ø,1}
			Match disk unit and controller with channel	({1,2},A,{Ø,1})
				({2,3},B,{Ø,1})

T5	XFER
	
	XFERWT
		{[{1,2},A,{Ø,1}],
		[{2,3},B,{Ø,1}]}
		{1,2,3}
			Match disk unit with disk request	([{1},*,*],1)
				([{2},*,*],2)
				([{3},*,*],3)

TOKEN FIRING TEMPLATES FOR DISK I/O
SUBSYSTEM MODEL (SERIAL PATH ALLOCATION)

TABLE 6-3



Marked TEPN Model of 2-Controller, 3-Disk Subsystem (Serial Path Allocation) with Firing Templates

Figure 6-6

DSKIONET = (P, T, I, O)

where,

P = {CHNAVL, DSKREQ, CNTWT, CNTAVL, DSKWT, DSKAVL, DSKSEK, XFER, XFERCP, XFERWT}

T = {T1, T2, T3, T4, T5}

I (T1) = {DSKREQ, CHNAVL}	O (T1) = {CNTWT, XFERWT}
I (T2) = {CNTWT, CNTAVL}	O (T2) = {DSKWT}}
I (T3) = {DSKWT, DSKAVL, CNTAVL}	O (T3) = {DSKSEK, CHNAVL}
I (T4) = {DSKSEK, CHNAVL}	O (T4) = {XFER}
I (T5) = {XFER, XFERWT}	O (T5) = {CNTAVL, DSKAVL, XFERCP, CHNAVL}

TEPN Model Structural Definition
Disk I/O Subsystem with Serial Path Allocation

Table 6-4

Token Attributes and Transition Token Templates. Token attributes and, consequently, token templates are defined by the specifications of the TEPN place and transition firing template attributes. More specifically, the token functional attribute set is derived primarly from the set of functions necessary for computation of the token delay time mapping (TDTM) value for each place. These functions can be identified in Tables 6-5 and 6-6 while the token type attribute is derived from the definition of the transition firing templates, defined in Table 6-3. In addition to the token type and functional attributes required of a token at its resident place, many tokens are defined to carry additional attributes which are required by other places later in the network. These attributes, which are defined as members of the token functional attribute set, could be viewed as "messages" in the process of being relayed from thrir source to their destination. An example of this is the attribute "SIZ" which (from Table 6-5) is required to compute the data transfer time for each request, shich is the TDTM value for a token residing at place XFER. The original source of this attribute is the token from place DSKREQ, where each disk request is initiated. In order to insure that this piece of information is available at place XFER when it is needed, a "functional attribute transfer path" is established and the attribute is passed along from token to token until it "arrives" with the appropriate token at XFER.

		PAQ			PEQ		
P-Name	Clock	QDisc	QBND	TDTM	QDisc	QBMD	PPF Set
DSKREQ	A[1]	FCFS[2]	*	$D=F_1(tok)$[5]	MFCFS[4]	*	TPUT[8]
CNTWT	A	FCFS[2]	*	D=∅	MFCFS	*	TPUTR[9]
CNTAVL	A	IP[3]	2	D=∅	IP	2	UTIL[10]
DSKWT	A	FCFS[2]	*	D=∅	MFCFS	*	TPUTR
DSKAVL	A	IP[3]	3	D=∅	IP	3	UTIL
DSKSEK	A	IP[3]	3	$D=F_2(tok)$[6]	IP	3	-
XFER	A	IP[3]	3	$D=F_3(tok)$[7]	IP	3	-
XFERWT	A	FCFS[2]	*	D=∅	MFCFS	*	TWT[11]
XFERCP	A	IP[3]	*	D=∅	IP	*	TPUT

1 Active
2 First-Come-First-Served
3 Infinite Processor
4 Multiple FCFS
5 F_1(tok) = INTARR, inter-arrival time function
6 F_2(tok) = ABS (TRK-Pas) * SEKTIM
7 F_1(tok) = SIZ * RATE
8 TPUT = total tokens that have passed place, by token type
9 TPUTR = TPUT per unit time (TPUT rate)
10 UTIL = utilization of place by waiting tokens, by type
11 TWT = total wait time from time token arrives at place to when token leaves

TEPN Place Attributes
TEPN Model of Disk Subsystem with Serial Path Allocation
Table 6-5

FUNCTION	MNEMONIC	DESCRIPTION
Request Inter-Arrival Time	INTARR	Returns the amount of time prior to releasing the next token from place DSKREQ
Request Number	REQNUM	Returns unique identifier assigned to each new disk request transaction.
Data Transfer Size	SIZ	Returns number of units of data to be transferred to this transaction.
Data Track Address	TRK	Returns the disk unit track address of the data to be transferred.
Disk Position	POS	Returns the current track address of the disk unit read/write head.
Disk Seek Time	SEKTIM	Returns the disk track-to-track seek time.

TOKEN FUNCTIONAL ATTRIBUTES
DISK MODEL OF DISK I/O SUBSYSTEM

TABLE 6-6

The token definitions required to represent the disk I/O subsystem are presented in Table 6-7. Note that several of these definitions show tokens with "composite" token types. These types are actually the concatenation of several types and are used to preserve the identity of the resources allocated to each transaction as the model is executed. The notation used in this table, while new, is intended to be somewhat self-explanatory. All definitions employed standard set notation. In the case of composite token types, lists enclosed in square brackets are single type elements. Also, when one of the elements of the domain set is itself a set, the implication is that any of the "inner" set may be matched with any other elements in the composite type to form a "legal" ordered re-tuple.

The final information required to define the tokens is the transition output token templates. These are implied by the definitions of Tables 6-6 and 6-7 and are formally presented in Table 6-8. For brevity, the table utilizes a functional notation to indicate transfer of type and attribute information. The notation should be clear with the possible exception of the notation TYPE = TYn (P), where n is a digit and P is a place name. This indicates the particular element of the ordered n-tuple which forms a composite type of a token from place P. The particular type value referred to can be determined from Table 6-6.

PLACE	TOKEN TYPE DESCRIPTION	TOKEN TYPE DOMAIN	FUNCTIONAL ATTRIBUTE SET (FAS)
DSKREQ	Disk Unit Number (DSK)	{1,2,3}	{REQNUM, SIZ, TRK}
XFERWT	DSK	{1,2,3}	{REQNUM}
CNTAVL	Channel Number (CHN)	{0,1}	-null-
CNTWT	*CHN, DSK	*[{0,1}, {1,2,3}]	{REQNUM, SIZ, TRK}
CNTAVL	Controller Number (CNT)	{A,B}	-null-
DSKWT	*CHN, DSK, CNT	*{[{0,1},{1,2},A],[{0,1},{2,3},B]}	{REQNUM, SIZ, TRK, POS, SEKTIM}
DSKAVL	DSK	{1,2,3}	{POS, SEKTIM}
DSKSEK	DSK	{1,2,3}	{REQNUM, SIZ, TRK, POS, SEKTIM}
XFER	*DSK, CNT, CHN	*{[{1,2},A,{0,1}][{2,3},B,{0,1}]	{REQNUM, SIZ, TRK, POS, SEKTIM}
XFERCP	DSK	{1,2,3}	{REQNUM}

*Composite Token Type

PLACE TOKEN DEFINITIONS

TEPN MODEL OF DISK SUBSYSTEM

TABLE 6-7

TRANSITION	I(T)	OUTPUT	TRANSITION OUTPUT TOKEN TEMPLATE
T1	{DSK REQ, CHNAVL}	XFERWT	TYPE = TY(DSKREQ), FAS = {REQNUM(DSKREQ)}
		CNTWT	TYPE = [TY(CHNAVL), TY(DSKREQ)], FAS = FAS(DSKREQ)
T2	{CNTWT, CNTAVL}	DSKWT	TYPE = [TY(CNTWT), TY(CNTAVL)], FAS = PAS(CNTWT)
T3	{DSKWT, DSKAVL}	CHNAVL	TYPE = TY1(DSKWT), FAS = null
		DSKSEK	TYPE = TY(DSKAVL), FAS = FAS(DSEKWT) + FAS(DSKAVL)
T4	{DSKSEK CHNAVL}	XFER	TYPE = [TY(DSKSEK), TY(CNTAVL), TY(CHNAVL)], FAS = FAS(DSKSEK)
T5	{XFER, XFERWT}	XFERCP	TYPE = TY(XFERWT), FAS = {REQNUM}
		DSKAVL	TYPE = TY1(XFER), FAS = {POS = TRK(XFER), SEKTIM}
		CNTAVL	TYPE = TY2(XFER), FAS = null
		CHNAVL	TYPE = TY3(XFER), FAS = null

TRANSITION OUTPUT TOKEN TEMPLATES
TEPN MODEL OF DISK SUBSYSTEM WITH PARALLEL PATH ALLOCATION

TABLE 6-8

TEPN Model of the Disk I/O Subsystem with Parallel Path Allocation

In order to effectively model parallel path allocation, the model of Figure 6-7 must be altered to allow for a) the return of the controller to the available state for the CNTAVL place during the time period taken by a disk seek operation; and, b) the reallocation of a controller from CNTAVL prior to the commencement of the transfer operation but following the completion of the disk seek operation. The exact implementation of this required modification to the original model depends upon the level of detail required for the study. In most cases, for example, parallel path allocation could be accurately modeled with the simplifying assumptions that (1) the controller is automatically deallocated at the beginning of every disk seek operation, and, (2) once the disk seek operation is complete, the transfer will begin as soon as the controller token is available. Assumption 1 does not take into account the case in which there is no disk seek time. In this case, the algorithm would probably be defined such that the controller would not be deallocated and the transfer operation would be allowed to begin immediately. This assumption, however, is not considered significant except in cases where very minute detail of modeling is necessary, since this assumption would be expected to have little impact upon the system performance. Furthermore, in the case where the disk seek is indeed zero, one could assume that transition T-4 would immediately fire, thus causing the immediate reallocation of the lost controller. The second assumption is certainly the more significant of the two assumptions in that it ignores the extra latency or rotational delay time caused when a controller is not immediately available to service a disk drive that has just completed a seek operation.[10] Therefore, in a system which is extremely busy, in terms of request volume, and which has many disk units attached to the same controller, this assumption could result in inaccuracy of the results. In cases where the above assumptions are considered unacceptable, a model may be further modified to represent the interactions at even the most detailed level. The more complex model is not included in this Thesis as it is not required to illustrate any special capabilities of the TEPN model.

Differences Between the Serial and Parallel Model

If the two assumptions mentioned in the last section are made, only one strucural modification is required to convert the model into an accurate representation of the disk subsystem with parallel allocation. This modification is to add an output arc from transition T1 to places CNTAVL and a corresponding input arc to transition T4 from place CNTAVL.

---

[10]Addition of an automatic "average latency time" to the TDTM computed transfer time would only partially solve this problem since it would be based upon an assumption of a known pattern of latency delays which may not be the case.

This modification cause the controller to be returned to the CNTAVL pool until it is required by another (or the same, at later time) transaction. The new marked structure is illustrated in figure 6-7. The TEPN "structure definition" is also presented, in table 6-9. This structural change requires a slight redefinition of the transition firing and token templates. The new definitions are shown in tables 6-10 and 6-11.

Execution of the Disk Subsystem Model

Figures 6-8 thorugh 6-13 illustrates an execution sequence of the model of figure 6-7 in which a disk request is traced through its "life cycle" within the network. Although the diagrams are somwhat self-explanatory, additional notes are included to help the reader follow the example. For best understanding, it is suggeted that the tables prensented earlier for the parallel path allocation disk subsystem model be followed closely through the execution cycle. Table 6-12 presents initial valves for each of the token functions to be used in TDTM computations. The cycle is broken down into six steps, as follows:

1. New Request Arrives (fig 6-8) at place DSKREQ, specifying the dirve required (DSKNUM), (which is the token's type attribute) data track address (TRK), the amounnt of data (SIZ), and a request number (REQNUMB). Example attributes are shown in table 6-12.

DSKIONET = (P, T, I, O)

where,

P = {CHNAVL, DSKREQ, CNTWT, CNTAVL, DSKWT, DSKAVL, DSKSEK, XFER, XFERCP, XFERWT}

T = {T1, T2, T3, T4, T5}

I (T1) =	{DSKREQ, CHNAVL}	O (T1) =	{CNTWT, XFERWT}
I (T2) =	{CNTWT, CNTAVL}	O (T2) =	{DSKWT}
I (T3) =	{DSKWT, DSKAVL}	O (T3) =	{DSKSEK, CNTAVL, CHNAVL}
I (T4) =	{DSKSEK, CHNALL}	O (T4) =	{XFER}
I (T5) =	{XFER, XFERWT}	O (T5) =	{CNTAVL, DSKAVL, XFERCP, CHNAVL}

TEPN Model Structural Definition
Disk I/O Subsystem with Parallel Path Allocation

Table 6-9

TRANSITION	INPUT PLACES	TOKEN TYPE DOMAIN	FIRING TEMPLATE GOAL	ENABLING FIRING TEMPLATES

T1	DSKREQ
	CHNAVL
		{ 1,2,3 }
		{ Ø,1 }
			Match any channel with disk request	({1,2,3}, {Ø,1})

T2	CNTWT
	CNTAVL
		{ [{Ø,1}, {1,2,3}] }
		{A,B}
			Match disk request with proper controller	[*,{1,2}],A)
				([*,{2,3}],B)

T3	DSKWT
	
	DSKAVL
		{[{Ø,1},{1,2},A],
		[{Ø,1}, {1,3},B]}
		{1,2,3}
			Match disk request with disk unit	([*,1,*],1)
				([*,1,*],2)
				([*,3,*],3)

T4	DSKSEK
	CNTAVL
	CHNAVL
		{1,2,3}
		{A,B}
		{Ø,1}
			Match disk unit with controller and channel	({1,2},A,{Ø,1})
				({2,3},B,{Ø,1})

T5	XFER
	
	XFERWT
		{[{1,2},A,{Ø,1}],
		[{2,3},B,{Ø,1}]}
		{1,2,3}
			Match disk unit with disk request	([{1},*,*],1)
				([{2},*,8],2)
				([{3},*,*],3)

TOKEN FIRING TEMPLATES FOR DISK I/O
SUBSYSTEM MODEL (PARALLEL PATH ALLOCATION)

TABLE 6-10

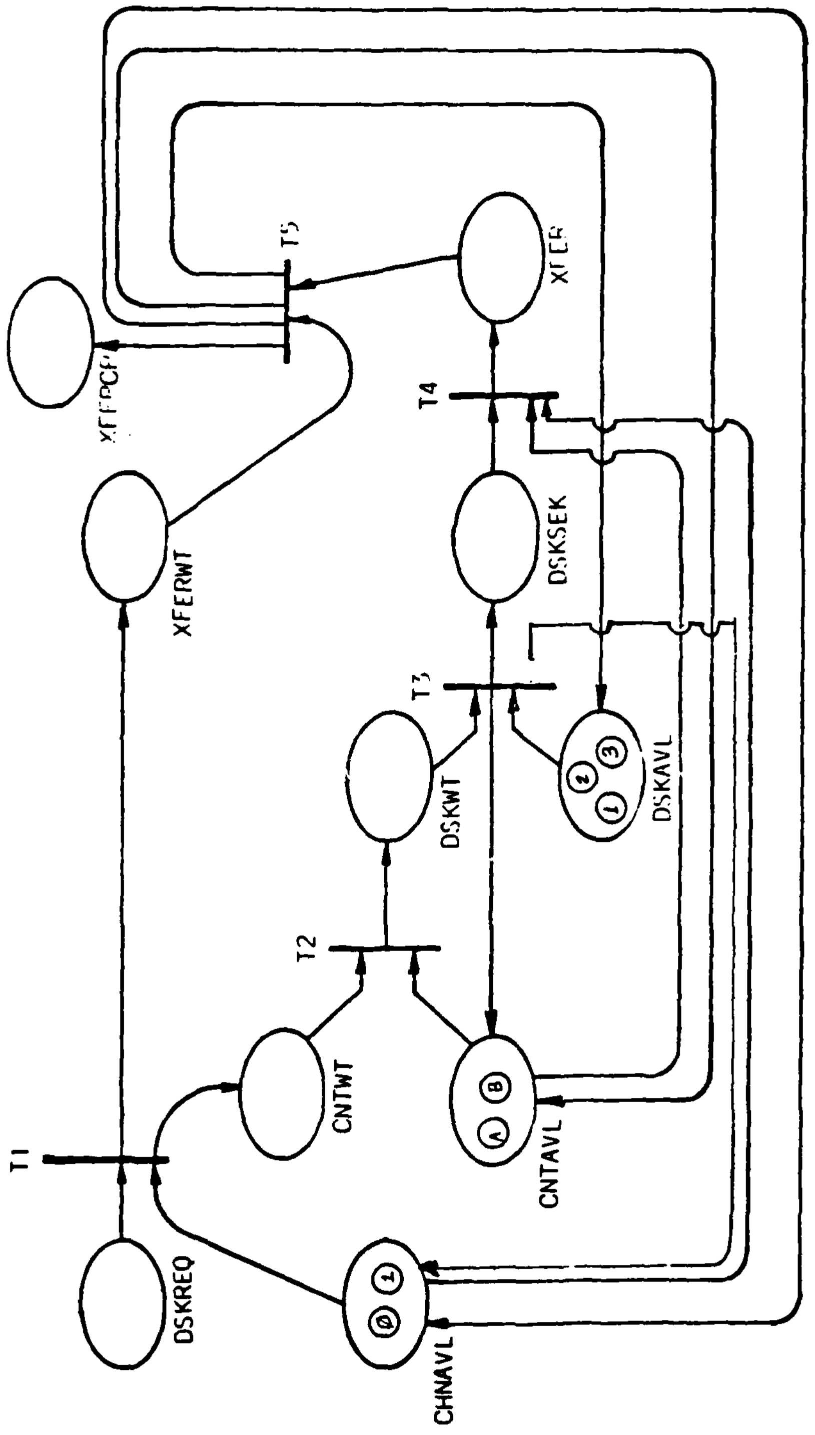
TRANSITION	I(T)	OUTPUT	TRANSITION OUTPUT TOKEN TEMPLATE
T1	{DSK REQ, CHNAVL}	XFERWT	TYPE = TY(DSKREQ), FAS = {REQNUM(DSKREQ)}
		CNTWT	TYPE = [TY(CHNAVL), TY(DSKREQ)], FAS = FAS(DSKREQ)
T2	{CNTWT, CNTAVL}	DSKWT	TYPE = [TY(CNTWT), TY(CNTAVL)], FAS = PAS(CNTWT)
T3	{DSKWT, DSKAVL}	CHNAVL	TYPE = TY1(DSKWT), FAS = null
		CNTAVL	TYPE = TY3(DSKWT), FAS = null
		DSKSEK	TYPE = TY(DSKAVL), FAS = FAS(DSEKWT) + FAS(DSKAVL)
T4	{DSKSEK, CNTAVL, CHNAVL}	XFER	TYPE = [TY(DSKSEK), TY(CNTAVL), TY(CHNAVL)], FAS = FAS(DSKSEK)
T5	{XFER, XFERWT}	XFERCP	TYPE = TY(XFERWT), FAS = {REQNUM}
		DSKAVL	TYPE = TY1(XFER), FAS = {POS = TRK(XFER), SEKTIM}
		CNTAVL	TYPE = TY2(XFER), FAS = null
		CHNAVL	TYPE = TY3(XFER), FAS = null

TRANSITION OUTPUT TOKEN TEMPLATES
TEPN MODEL OF DISK SUBSYSTEM WITH PARALLEL PATH ALLOCATION

TABLE 6-11

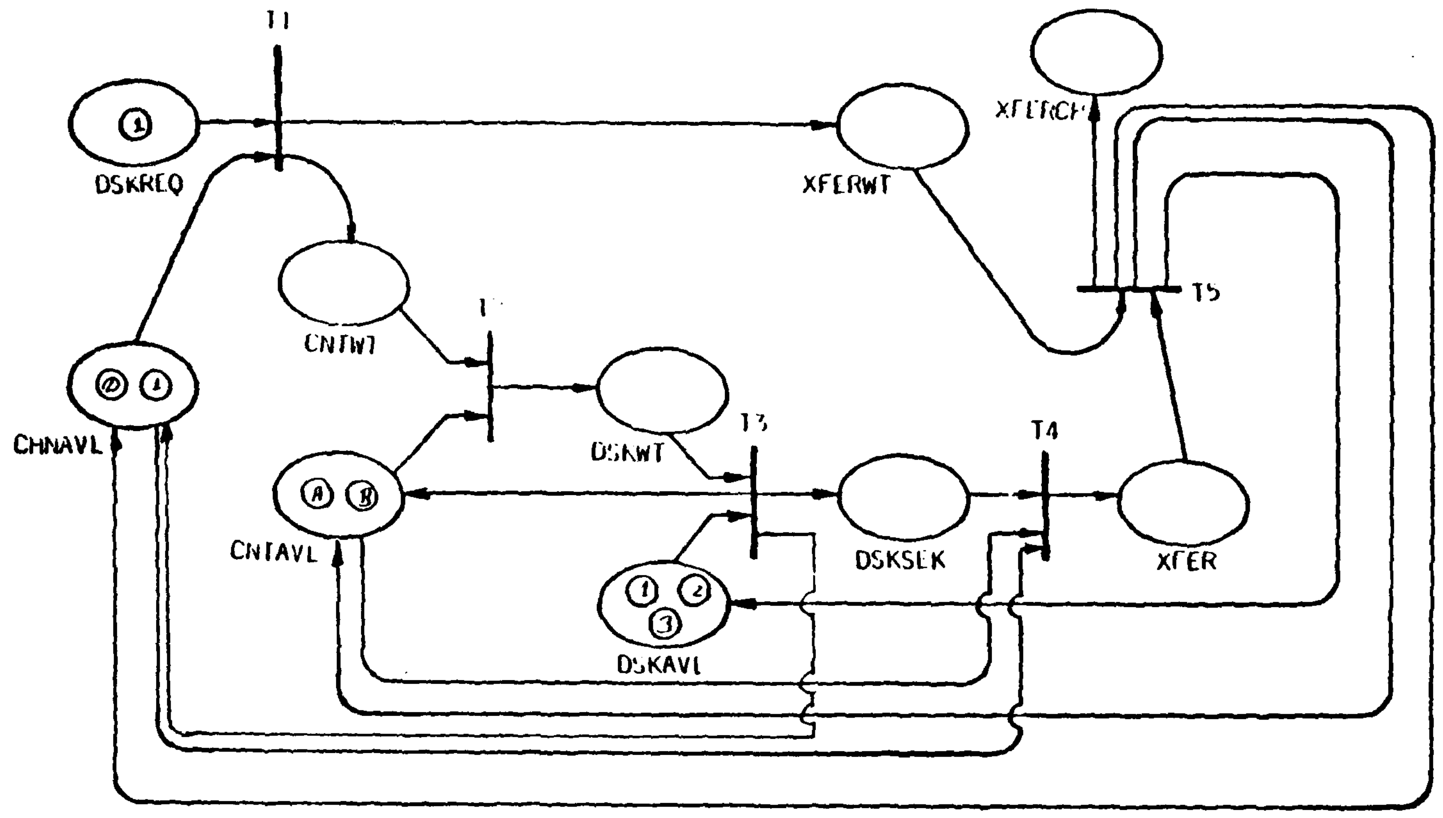
PLACE	TOKEN TYPE	TOKEN FAS
CHNAVL	Ø	null
	1	null
CNTAVL	A	null
	B	null
DSKAVL	1	pos=180, SEKTIM=1 ms, RATE=.5 ms
	2	pos=100, SEKTIM=1 ms, RATE=.5 ms
	3	pos=200, SEKTIM=1 ms, RATE=.5 ms
DSKREQ	1	REQNUM=1, TRK=180, SIZ=50

INITIAL VALUES OF TOKEN ATTRIBUTES
TABLE 6-12



MARKED TEPN MODEL OF DISK SUBSYSTEM WITH PARALLEL PATH ALLOCATION

FIGURE 6-7



MARKED TEPN MODEL AFTER DISK REQUEST ARRIVAL

FIGURE 6-8

2. Channel Allocation (fig 6-9) of a channel to the request.

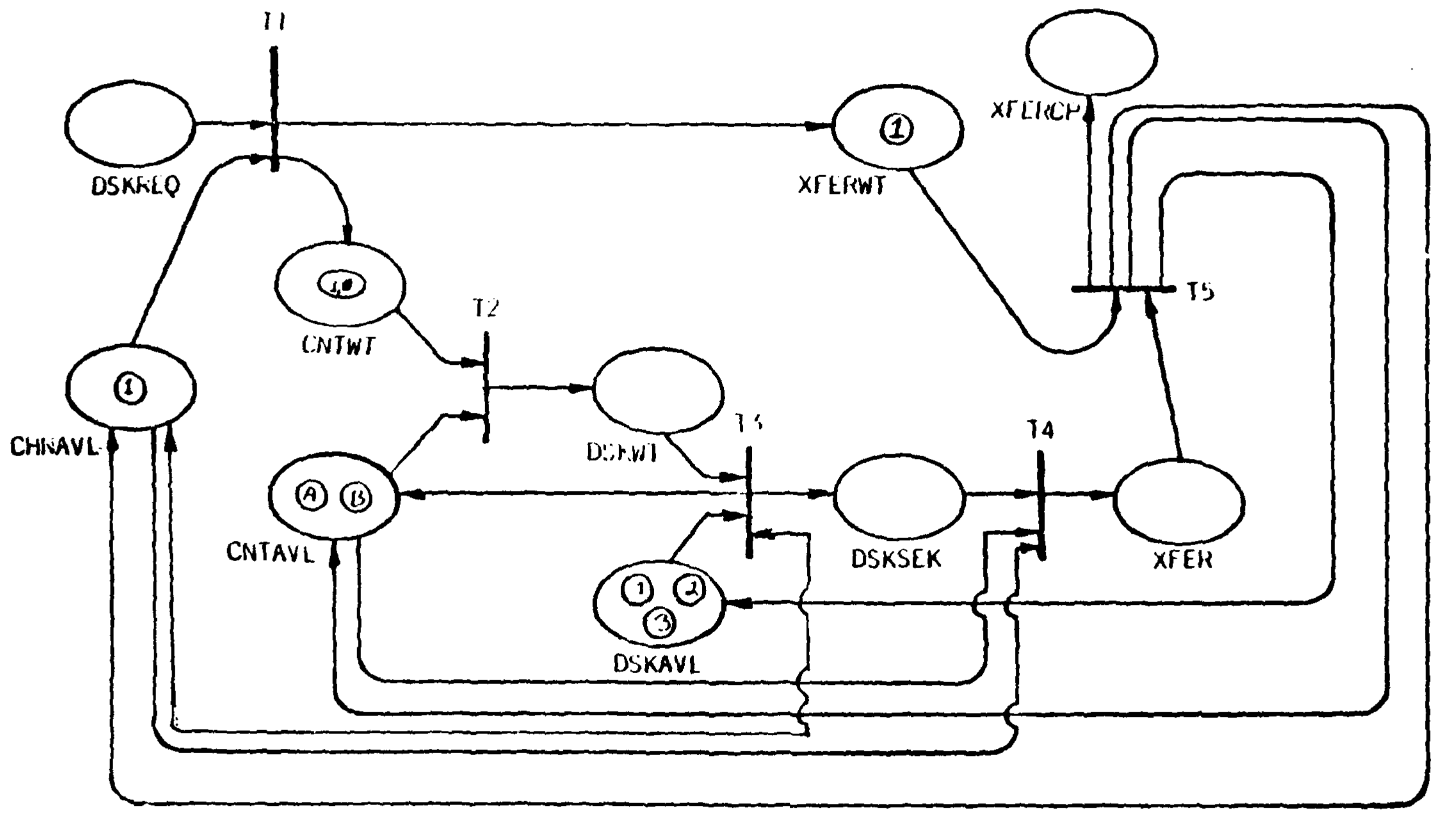
3. Controller Allocation (fig 6-10) of controller "A" from place CNTAVL. Because of the token template defined for transition T2, only controller "A' can be used in combination with DSKNUMB "1" to enable T2. Note that the type of the output token from T2 to ONKWT is a composite type indicating the enitre channel-controller-disk path being utilized.

4. Disk Allocation / Release of Controller (fig 6-11) of disk "1" and controller "A'. Because the parallel path algorithm, the controller is not retained during the disk seek time.

5. Disk Seek (fig 6-12) according ot the pre-defined TDTM in which WAIT = SEKTIME * abs(TRK-POS). =1ms *(180-100) =80ms, using values from Table 6-12.

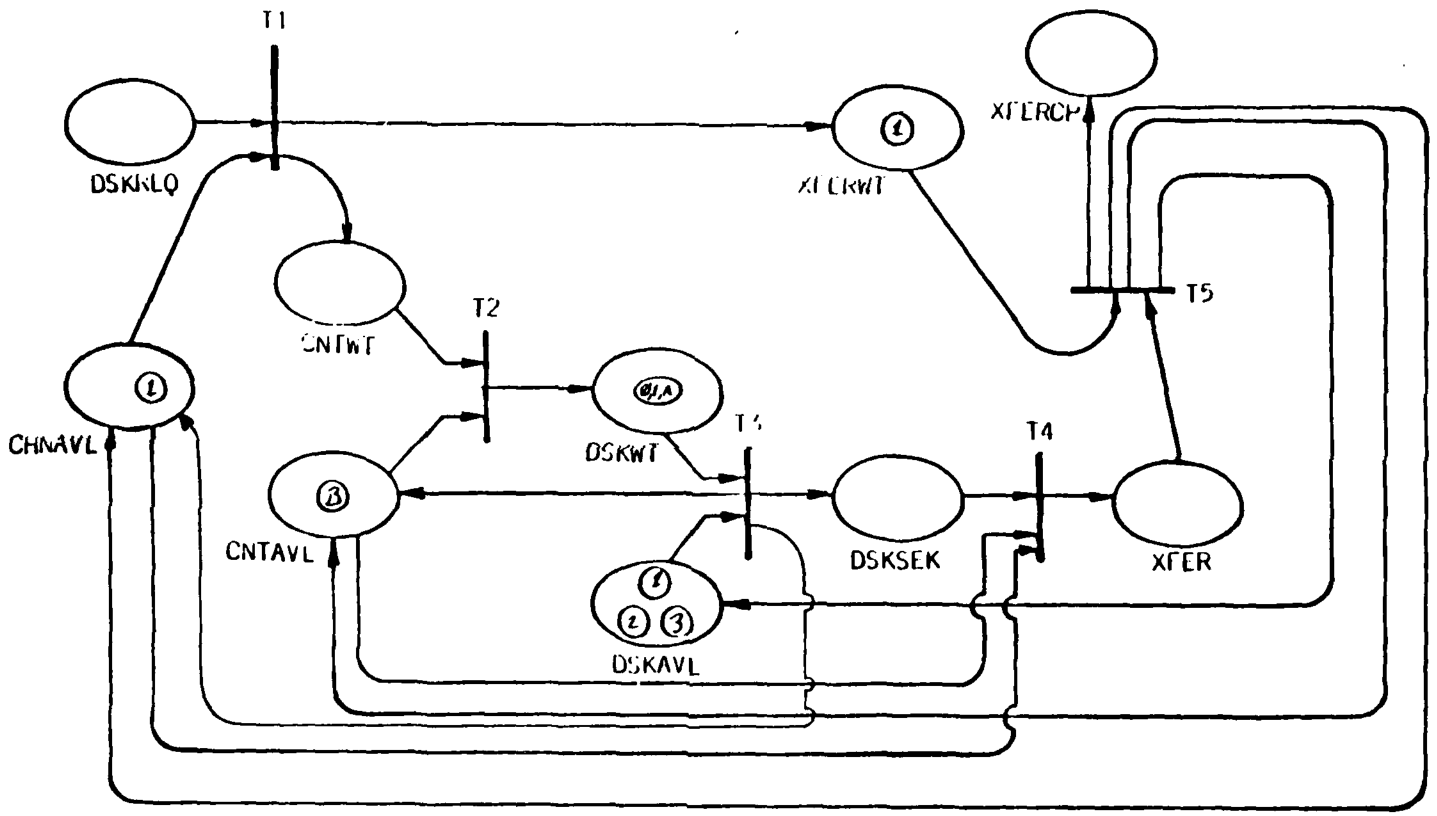
6. Transfer Begins (fig 6-12) and controller and channel are both reallocated to the data transfer request. The transfer time is computed according to the TDTM defined for place XFER. In this case the wait time would be: SIZ * RATE = 50) block =25 ms.

7. Transfer Complete (fig 6-13) and all resources deallocatid. Note merger at transition T5 with place XFERWT. This place is only for recording the total wait time fro request processing, which for this example would be total of the wait time, or: 50 ms + 80 ms= 130 ms.



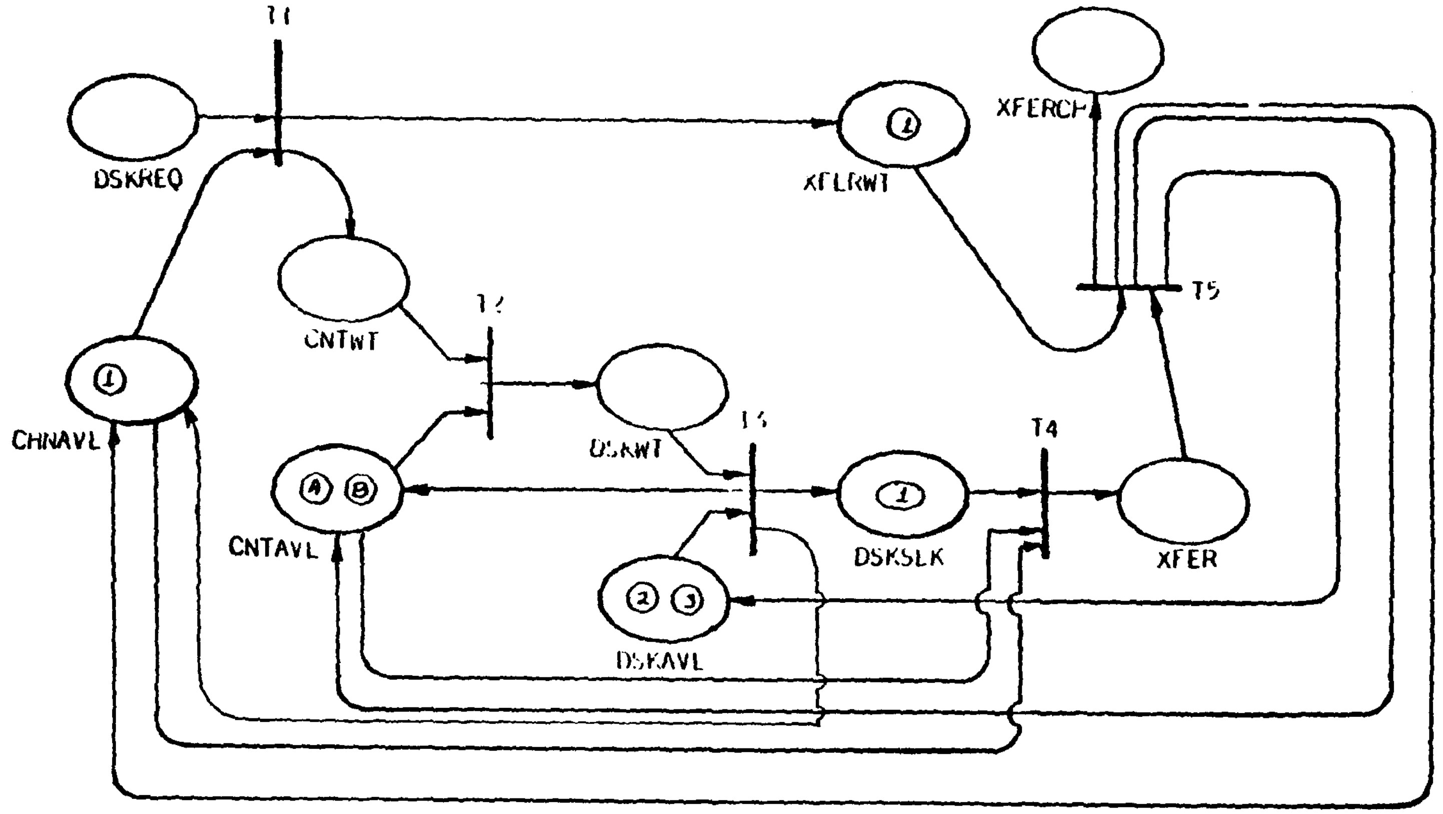
MARKED TEPN MODEL AFTER TRANSITION T1 FIRES
INDICATING ALLOCATION OF CHANNEL TO INCOMING DISK REQUEST

FIGURE 6-9



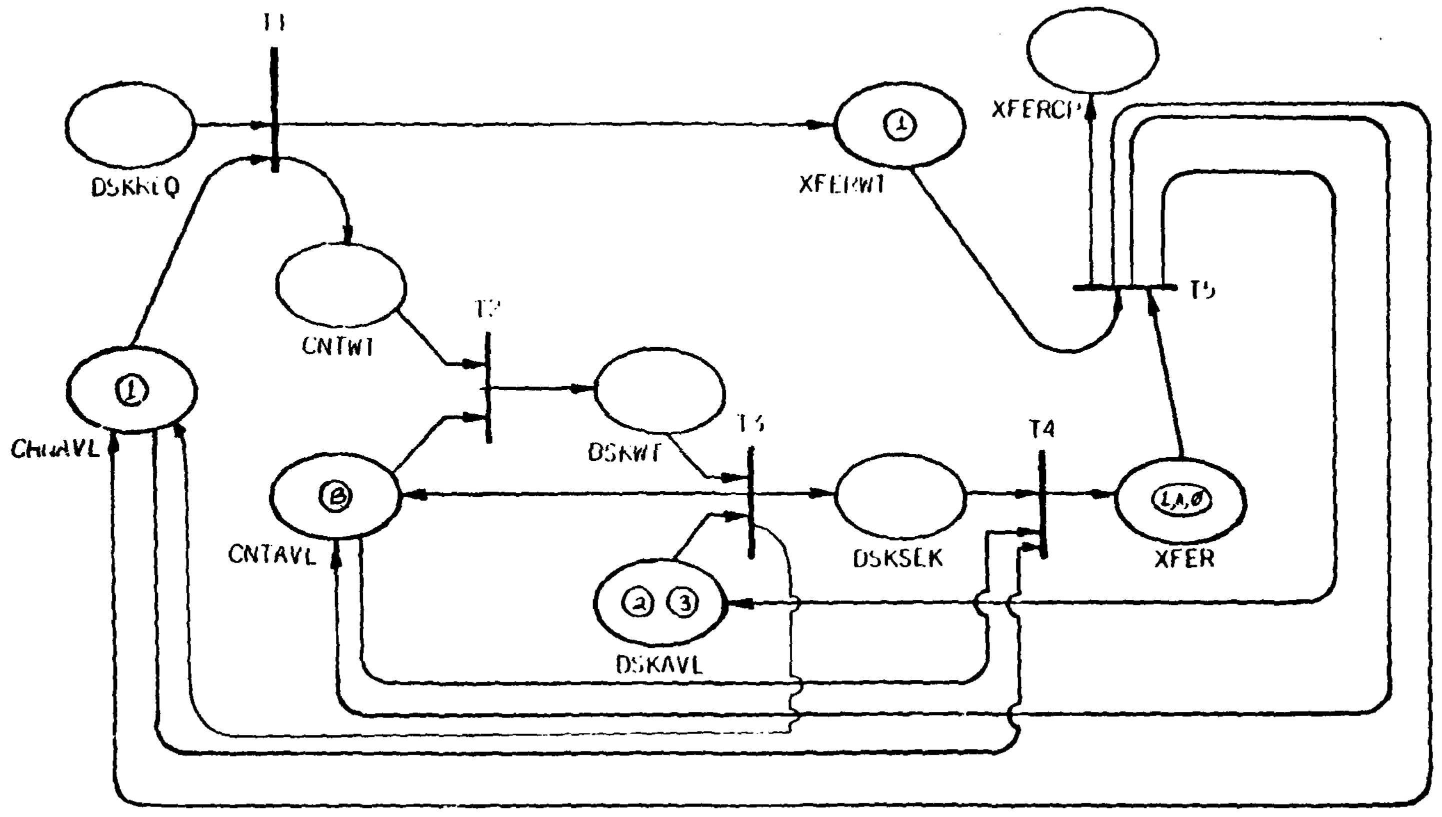
MARKED TEPN MODEL AFTER TRANSITION T2 FIRES
INDICATING ALLOCATION OF CONTROLLER TO DISK REQUEST

FIGURE 6-10



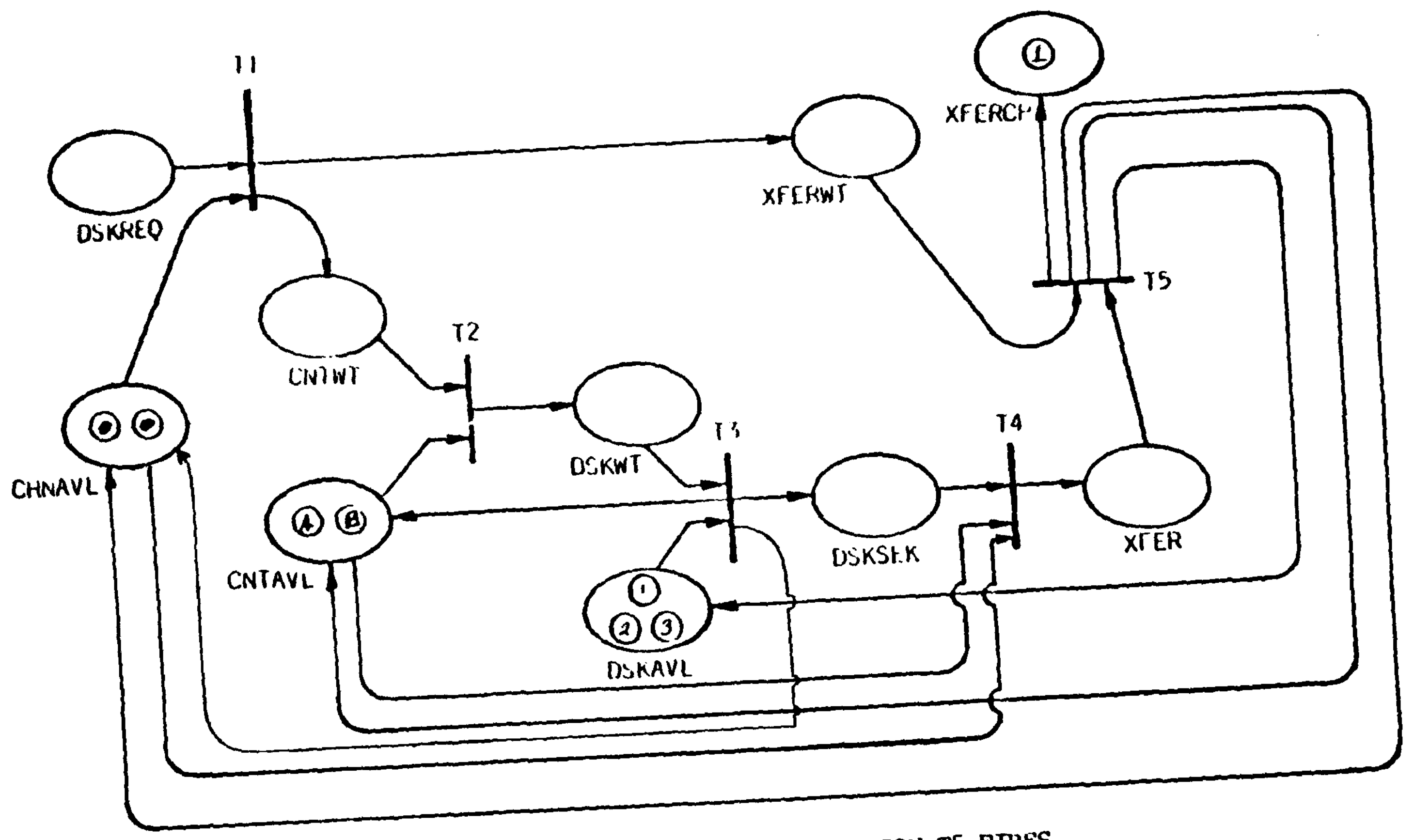
MARKED TEPN MODEL AFTER TRANSITION T3 FIRES
INDICATING ALLOCATION OF REQUESTED DISK UNIT TO DISK REQUEST

FIGURE 6-11



MARKED TEPN MODEL AFTER TRANSITION T4 FIRES
INDICATING COMPLETION OF DISK SEEK OPERATION

FIGURE 6-12



MARKED TEPN MODEL AFTER TRANSITION T5 FIRES
INDICATING COMPLETION OF DISK REQUEST TRANSACTION

FIGURE 6-13

Further analysis of the model and experimentation with various initial markings will quickly reveal the versatility and the structural integrity of this model, and of the TEPN model in general.

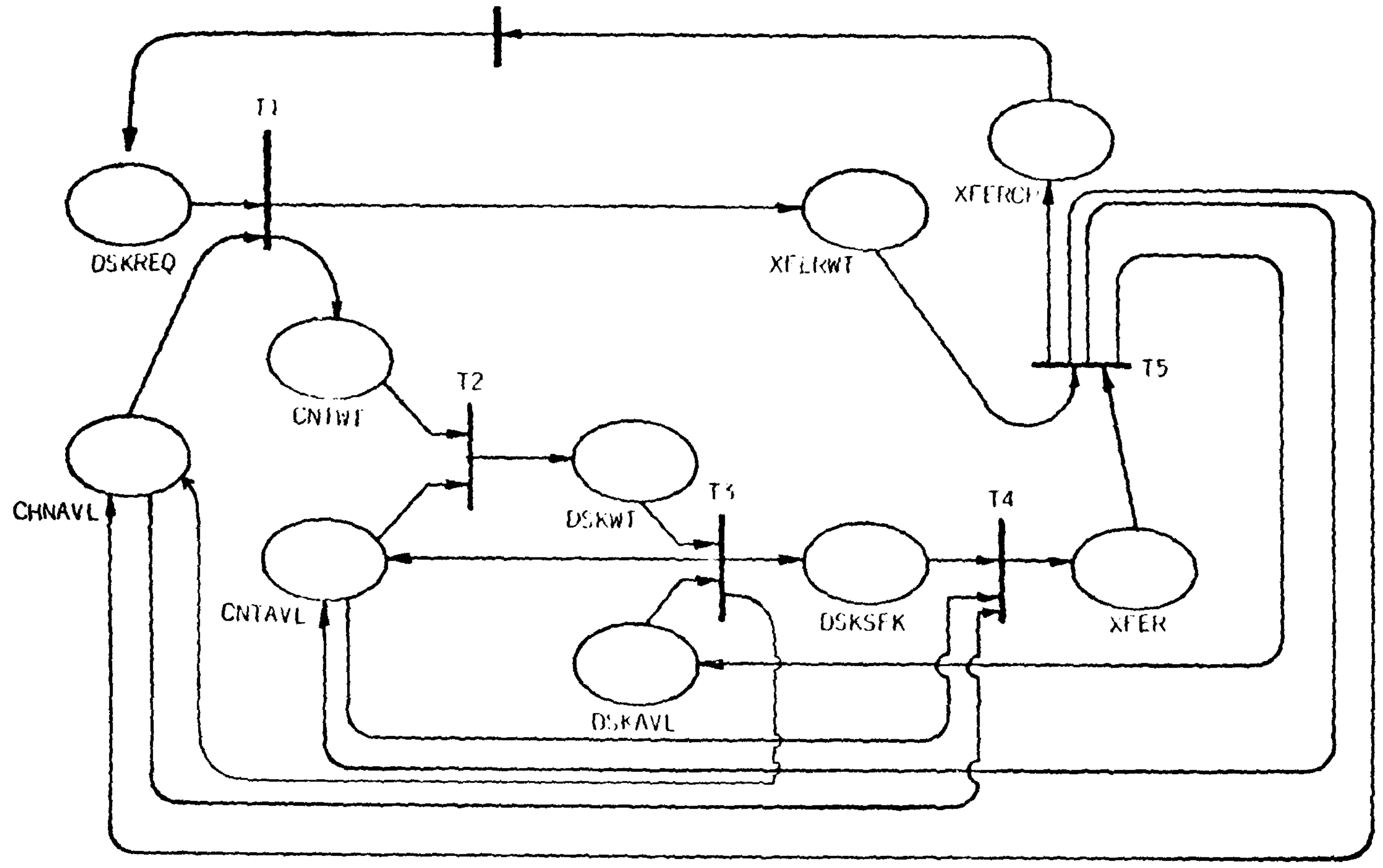
Token Generation. The final question that will be discussed in this Thesis concerning the example TEPN model is that of devising a method for continuous "generation" of input tokens, such as the tokens which would come to the DSKREQ place in order to evaluate the subsystem performance. While an in-depth discussion of this topic is not appropriate at this point, a brief presentation of some of the natural alternatives offered by the TEPN is again illustrative of the potential that the methodology has for CPE modeling. In particular, we identify three ways that the TEPN can handle this need:

1. Closed Network

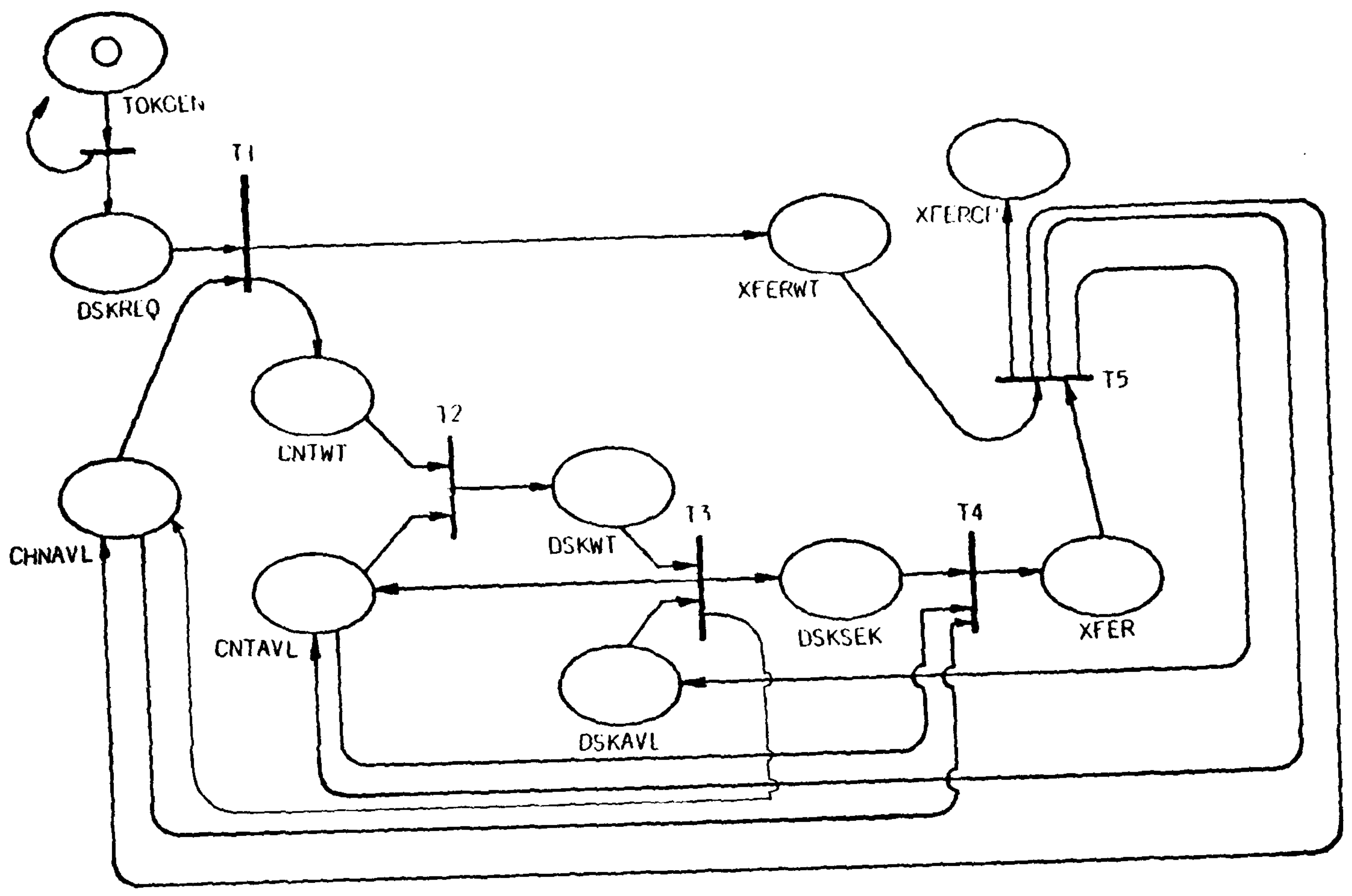
By drawing an arc from transition T5 to place DSKREQ, figure 6-6 can be transformed into a closed system which will continue to "recycle" its "workload" indefinitely. In this case, the parameters of DSKREQ tokens could be either drawn from a random field or kept constant (to represent, for example, known "typical" requests). Figure 6-14 illustrates the resulting structure.

2. Open Network/Randomly Driven

Figure 6-15 illustrates a second implementation method in which a special "token generator" place has been added. The net would be initialized by inserting a token at place TOKGEN. After some deter-



Disk Subsystem Model
Closed Network Implementation
Figure 6-14

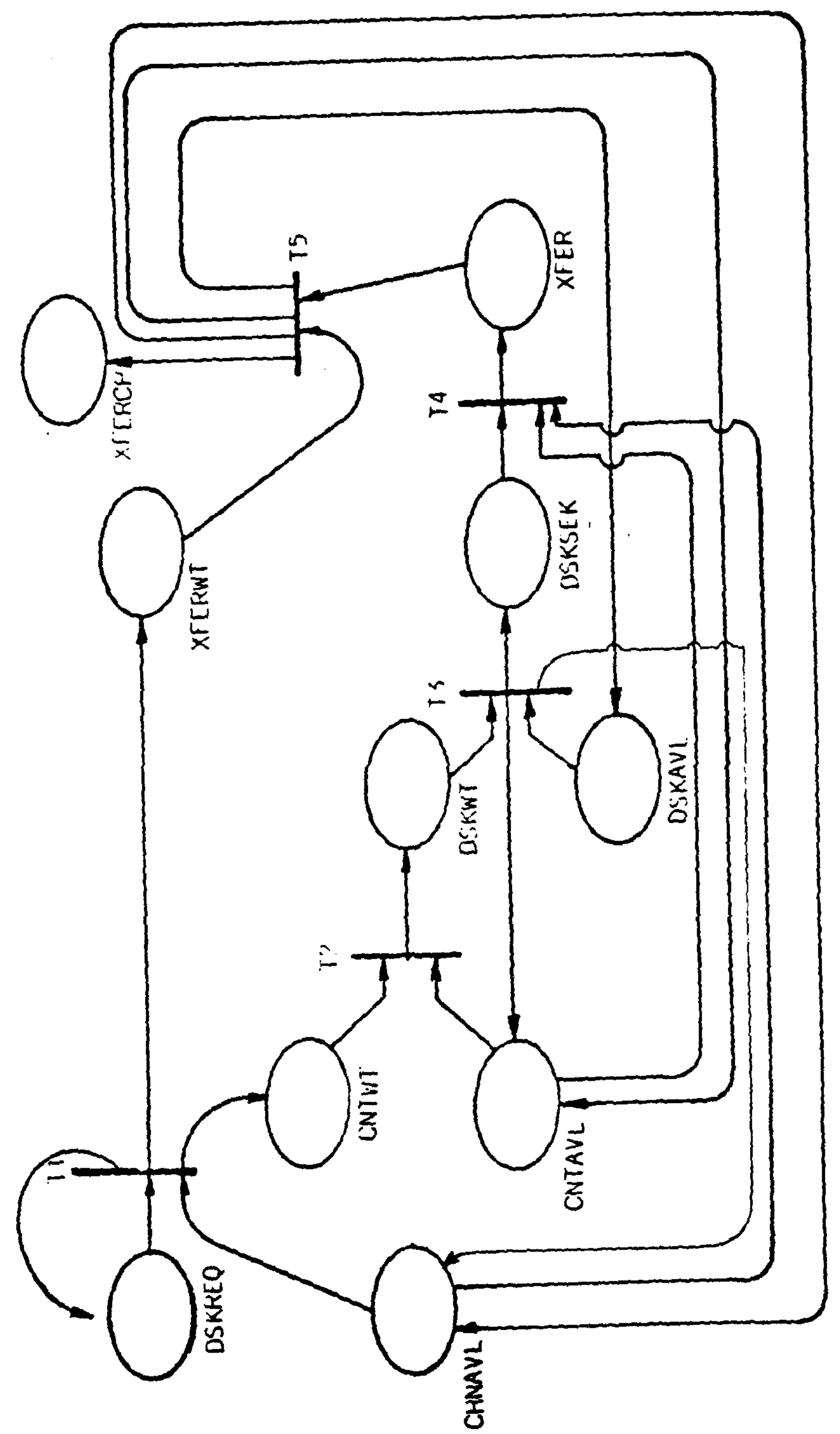


Disk Subsystem Model
Open Network / Randomly Driven Implementation
Figure 6-15

mined delay time, transition T would fire, sending a token to DSKREQ and returning a token to TOKGEN. The token template could be designed to either retain standard parameters or to chose new parameters either randomly or from a separate source.

3. Open Network/Trace Driven

The model of Figure 6-8, finally, could be used as a trace driven model by simply defining the token template processor such that a "READ ATTRIBUTE" operation used to obtain an attribute value for inclusion in a new token could be an input from any source, including a file of actual system transactions. Note that this implementation does not require any modification to either the TEPN Definition or to the implementation specifications. This implementation is illustrated in Figure 6-16.



Disk Subsystem Model
Open Network / Trace Driven Implementation
Figure 6-16

CHAPTER VII

SUMMARY AND CONCLUSIONS

The primary result of this research is the definition of a new CPE modeling structure, the Time-Extended Petri Net, which effectively retains logical synchronization and concurrency characteristics of systems.

Cost effectiveness is an important consideration when evaluating the potential of any CPE methodology. A further result of this research, therefore, is the demonstration of the feasibility to (1) develop an efficient computerized TEPN modeling system, and (2) use the TEPN in analyzing practical CPE problems. The initial, though limited results included in this thesis demonstrate that the TEPN is applicable to at least a small class of problems. More research and testing will be required before the TEPN can be declared to be a general CPE tool; this research is underway at this time.

At this point, the use of the TEPN as a modeling tool for many problems is still a cumbersome task. There is still considerable work ahead before the implementation becomes efficient and sufficiently user oriented to allow cost effective modeling of large system evaluation problems. However, the results of this Thesis seem to indicate the potential for building and evaluating effective models of such systems at a fraction of the cost of building full-scale simulation models.

Appendix A

Decomposition of a TEPN Model into a Petri Net

One of the stated advantages of the TEPN structure is that it has the flexibility to "decompose" into a Petri net without any alteration of the basic model structure. Therefore, the same model could be used to study many theoretical properties (such as state reachability) using the identical model.

Building a "default" Petri net consists of steps:

(1) Define all places within the net with the standard default parameters;

(2) Define all transition firing and token

TEPN Place Default Parameters

On an operational level, the TEPN and Petri net place differ primarily in that (a) "time" is resolved within the place, and (b) the TEPN place is capable of ordering its resident tokens and using this ordering to control the exit of tokens to its output transitions. In order to decompose a TEPN place so that it is operationally equivalent to the standard Petri net place, therefore, one must define all attributes such that the above two differences are eliminated. Such a set of parameters is shown in table A-1.

The net effect of the default attributes is to:

(1) remove time resolution from the place by "deactivating" the clock to a constant zero state,

(2) remove the concept of "token delay time" by defining the TDTM as a constant zero function,

(3) define the place marking as a "bag" or set of tokens rather than a queue or sequence by defining a queueing discipline which has an infinite number of processors resulting in elimination of any queueing delays, and,

(4) define the place as "unbounded," in accordance with the generalized Petri net definition.

If all places in the structure of figure 6-6 were defined with default attributes and if transition firing templates were left undefined, the net "performances" would completely revert to a Petri net (although the internal structure would still be a TEPN, as evidenced by the retention of a set of place performance functions.

<u>TEPN Transition Default Parameters Default Transition Firing Templates</u>

The Petri net transition has no mechanism for controlling token flow other than the standard enabling rules. Therefore, the TEPN transition default parameters must be defined such that the enabling and firing rules conform to the standard: if a token exists at each input places, regardless of the tokens' attributes. Therefore, the default transition parameter set includes a <u>null firing template</u> conforming to definition A.1, below.

Definition A-1. Null TEPN Transition Firing Template

A TEPN transition firing template is said to be <u>null</u> if the transition will be enabled by any arbitrary combination of input tokens as long as there is at least one token at each input place. In functional terms, for any transition T,

TFT(T) = "ENABLED" if $LIMi > 0$ for every Pi in I(T)

where, LIMi is the local internal marking of place Pi.

<u>Default TEPN Token and Transition Token Template</u>

The only TEPN token attribute which is supported by the Petri net is the token type , or "color". Therefore, all token templates within a default transition are restricted to defining a simple colored token. As with the TEPN, however, there is still considerable flexibility as to how the color is determined.

The default TEPN token is defined by definition A-2. Based upon this definition, the TEPN Transition Token Template will only be concerned with they token type or color.

Definition A-2. Default TEPN Token

The default TEPN Token is a TEPN token with a null token Functional Attribute Set (FAS).

Attribute	Value	Meaning/Remarks
CLOCK	INACTIVE	Place clock is a constant zero
PAQ Queue Discipline	IP	"Infinite Processor" Queue disciplines
PAQ Queue Bound	∞	Infinite Queue Bound
PAQ TDTM	D = FØ(token)	Constant zero delay time function
PEQ Queue Discipline	IP	"Infinite Processor" discipline
PEQ Queue Bound	∞	Infinite Queue Bound
Place Performance Functions	TPUT	Total number of tokens throughput
	QMAX	The largest queue size attained during run
	QSIZE	Average size of queue from sample

Place Attributes of "Default" Place
Table A-1

BIBLIOGRAPHY

Alexander, W.P., III. "Analysis of Sequencing in Computer Programs and Systems," Department of Computer Sciences technical Report TR35, PhD Dissertation, The University of Texas, Austin, Texas, August 1974.

Anderson, J.W. "Primitive Process Level Modeling and Simulation of a Multiprocessing Computer System," PhD Dissertation, Department of Computer Sciences, The University of Texas, Austin, Texas, May 1974.

Baer, J.L. "A Survey of Some Theoretical Aspects of Multiprocessing," Computing Surveys, Volume 5, Number 1, (March 1973), pp. 31-80.

Baer, J.L. "Modeling for Parallel Computation: A Case Study," in Proceedings of the 1973 Sagamore Computer Conference on Parallel Processing, Syracuse University, (August 1973).

Baskett, Forest, III. "Mathematical Models of Multiprocessor Computer Systems," PhD Dissertation, Department of Computer Sciences, The University of Texas, Austin, Texas, December 1970.

Berztiss, A.T. Data Structures Theory and Practice. Academic Press, New York, 1971.

Bredt, T.H. Analysis of Parallel Systems, Stanford Electronics Laboratories TR-7, Stanford University, August 1970.

Brice, R. "A Study of Feedback Coupled Resource Allocation Policies in a Multiprocessing Computer Environment," Technical Note TSN-35, PhD Dissertation, Department of Computer Sciences, The University of Texas, Austin, Texas, August 1973.

Browne, J.C.: Chandy, K.M.; Brown, R.M.; Keller, T.W.; Towsky, D.F.; and Dissky, C.W. "Hierarchical Techniques for the Development of Realistic Models of Complex Computer Systems," Proceedings of the IEEE, Volume 63, Number 6, (June 1975).

Buzen, J.P. "Queueing Network Models of Multiprogramming," PhD Dissertation, Harvard University, Cambridge, Massachusetts, August 1971.

Case User's Manual, TESDATA Systems Corporation, McLean, VA, 1962.

Casstevens, B.J.B. "A Simulation and Analytical Modeling Package for the Design and Evaluation of Complex Computer Networks," Masters Thesis, The University of Texas, Austin, Texas, May 1975.

Commoner, F.; Holt, A. W.; Even, S.; and Pnueli, A. "Marked directed graphs," J. Computer and Systems Science, Volume 5, Number 9 (Oct. 1971), pp. 511-523.

Efron, R., and Gordon, G. "A general purpose digital simulator and examples of its application. Part I: Description of the simulator." IBM Syst. J., Volume 3, Number 1 (1964), pp. 22-34.

General Purpose Systems Simulator II. Form B20-6346, IBM Corp., White Plains, N. Y., 1963.

Holt, A.W., Saint, H., Shapiro, R.M. and Warshall, S. "Final Report of the Information System Theory Project", Technical Report RADC-TR-68-305, Rome Air Development Center, Griffis Air Force Base, New York, September 1968.

Howard, J.H. "Coordination of Multiple Processes in Computer Operating Systems," PhD Dissertation, Department of Computer Science Technical Note TSN-16, The University of Texas, Austin, Texas, December 1970.

Johnson, D.S. "A Process-Oriented Model of Resource Demands in Large Multiprocessing Computer Utilities." PhD Dissertation, Department of Computer Science, The University of Texas, Austin, Texas, August 1972.

Keller, R. M. "Formal verification of parallel programs," Comm. ACM, Volume 19, Number 7 (July 1976), pp. 371-384.

Keller, T.W. ASQ Manual, Department of Computer Sciences Technical Report TR-27, The University of Texas, Austin, Texas, October 1973.

Kiviat, P. J., Villanueva, R., and Markowitz, H. M. The SIMSCRIPT II Programming Language. Prentice-Hall, Englewood Cliffs, N. J., 1969.

Lien, Y. E. "Termination properties of generalized Petri nets," SIAM J. Computing, Volume 5, Number 2 (June 1976), 251-265.

Lucas, H.C. "Performance Evaluation and Monitoring," Computing Surveys, Volume 3, Number 3, (September 1971) pp. 79-91.

MacDougall, M.H. "Computer System Simulation: An Introduction," Computing Surveys, Volume 2, Number 3, (September 1970), pp. 191-209.

Nielson, N. R. "ECSS: An Extendable Computer System Simulator," Proc. Third Conf. on Applications of Simulation, Los Angeles, California, 1969, pp. 114-129 (ACM/AIIE/IEEE/SHARE/SCi/TIMS).

Noe, J.D. "A Petri Net model of the CDC 6400," Report 71-04-03, Computer Science Department, University of Washington, Seattle, Washington, 1971.

Noe, J.D., and Nutt, G.J. "Validation of a Trace-driven CDC 6400 Simulation," Proceedings Spring Joint Computer Conference (May 1972), pp. 749-758.

Noe, J.D. and Nutt, G.J. "Macro E-Nets for representation of parallel systems," IEEE Trans. Comp. Volume C-22, Number 8, (Aug. 1973), pp. 718-727.

Parnas, D.L. "A Technique for Software Module Specification with Examples" Comm.ACM, Volume 15, Number 5, (May 1972) pp. 330-336.

Peterson, J.L. "Modeling of parallel systems, " PhD Dissertation, Department of Electrical Engineering, Stanford University, Stanford, California, December 1973.

Peterson, J.L. "Petri Nets," Computing Surveys, Volume 9, Number 3, (September 1977).

Peterson, J.L. "Colored Petri Nets," personal correspondance and telephone conversations with F.B. Berlin (April, 1979).

Petri, C. A. "Kommunikation mit Automaten," Schriften des Rheinisch-Westfalischen Institutes fur Instrumentelle Mathematik an der Universiiat Bonn, Heft 2, Bonn, W. Germany 1962; translation: C. F. Greene, Supplement 1 to Tech. Rep. RADC-TR-65-337, Vol. 1, Rome Air Development Center, Griffiss Air Force Base, N. Y., 1965, 89 pp.

Ramchandani, C. "Analysis of asynchronous concurrent systems by timed Petri nets," PhD Dissertation, Department of Electrical Engineering, MIT, Cambridge, Massachusetts, 1974; also MAC TR 120, Project MAC, MIT, Cambridge, Massachusetts, (February 1974).

Rodriguez, J.E. "A Graph Model for Parallel Computations," Project MAC TR-64, PhD Dissertation, MIT, September 1969.

Schwetman, H.D. "A Study of Resource Utilization and Performance Evaluation of Large Scale Computer Systems." TSN-12, Computation Center, The University of Texas, Austin, Texas, July 1970.

Sherman, S.W. "Trace-Driven Modeling Studies of the Performance of Computer Systems," Computation Center TSN-30, PhD Dissertation, Department of Computer Science, The University of Texas, Austin, Texas, August 1972.

Sherman, S.W. and Browne, J.C. "Trace-driven modeling: Review and overview," Symposium on the Simulation of Computer Systems, Gaithersberg, Maryland, June 1973.

Sherman, S.W., Howard, H.H., Jr., and Browne, J.C., "Trace-Driven Studies of Deadlock Control and Job Scheduling." Lecture Notes in Computer Science (edited by G. Goos and J. Hartmanis), Springer-Verlag, New York, 1975.

VITA

Frank Brett Berlin was born in Minneapolis, Minnesota, on June 15, 1950, the son of Lorraine Stanley Berlin and William August Berlin. After completing high school at Vicenza American High School, in Vicenza, Italy, he entered the United States Air Force Academy in June, 1968. In 1972, he received the Bachelor of Science degree and was commissioned a second lieutenant in the United States Air Force. Since 1972, he has been employed as an Air Force Officer working in the field of computer science in various locations in the United States and abroad.

Permanent address: 7011 Isabelle Dr.
Austin, Texas

This thesis was typed by Becky Cunningham, Alicia Guice, Trudi Berlin, Susan Dakkon, Dawn Murphy, Jan Oppenheimer, and Kathi Nolen.

LME